Titre: Title:	Modélisation d'une application de décentralisation de serveurs web
Auteur: Author:	Thibaud Pierre Perret
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Perret, T. P. (2013). Modélisation d'une application de décentralisation de serveurs web [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1255/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1255/
Directeurs de recherche: Advisors:	Samuel Pierre
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

MODÉLISATION D'UNE APPLICATION DE DÉCENTRALISATION DE SERVEURS WEB

THIBAUD PIERRE PERRET
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
NOVEMBRE 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MODÉLISATION D'UNE APPLICATION DE DÉCENTRALISATION DE SERVEURS WEB

présenté par : PERRET Thibaud Pierre
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

Mme BELLAÏCHE Martine, Ph.D., présidente
M. PIERRE Samuel, Ph.D., membre et directeur de recherche
M. QUINTERO Alejandro, Doct., membre

À mes parents et
beaux-parents. . .

REMERCIEMENTS

Je tiens à remercier tout particulièrement mon directeur de recherche, le professeur Samuel Pierre pour son encadrement et ses conseils de recherche toujours pertinents qui m'ont guidé tout au long de ma maîtrise mais aussi pour ses conseils sur le plan professionnel. Je voudrais également en profiter pour le remercier de son soutien moral et financier qui m'ont aidé tout au long de mon projet.

Je souhaite saisir l'occasion de remercier mes parents et beaux-parents qui m'ont permis de faire des études longues dans mon domaine de prédilection et grâce auxquels j'ai pu traverser l'atlantique pour les continuer. Merci de votre patience et merci d'être là, à mes côtés.

Enfin, je tiens aussi à exprimer ma gratitude aux membres du LARIM pour leur bonne humeur et la bonne ambiance de travail. Le LARIM est un peu une deuxième maison.

Résumé

L'utilisation d'Internet dans notre vie est aujourd'hui devenue quotidienne et indispensable. Au delà de l'accès à des services en ligne gouvernementaux ou commerciaux, Internet a permis à ses utilisateurs de diffuser de l'information accessible par l'ensemble des autres utilisateurs. L'Internet est un réseau qui ne comprend structurellement aucun centre et qui n'introduit aucun hiérarchie parmi les machines connectées et disposant d'adresses IP. On assiste cependant à une asymétrie de plus en plus importante entre les différents opérateurs réseaux. Certains opérateurs Internet se spécialisent dans la fourniture d'accès Internet aux particuliers, alors que d'autres sont spécialisés dans l'hébergement de serveurs et possèdent la majorité du contenu désiré par les internautes. Cela entraîne donc un trafic inter-AS totalement déséquilibré, permettant difficilement à de nouveaux opérateurs de s'interconnecter, en plus de poser des problèmes de concentration de l'information auprès de seulement quelques très grosses entreprises, ce qui soulève des problèmes de vie privée et de contrôle de données numériques. De par les usages, des centres artificiels sont ainsi créés. Le protocole IP étant neutre, il ne différencie pas les paquets provenant de serveurs ou de clients et ne fait que les transporter dans le réseau jusqu'à leur destinataire. L'hébergement sur une connexion Internet à usage personnel est donc possible mais relativement limité puisque la bande passante de ces utilisateurs est relativement faible et ces serveurs supporteraient très mal la montée en charge.

Notre projet vise donc à proposer une manière simple et efficace pour un internaute quelconque de partager du contenu en s'affranchissant d'un hébergement en salle machine sauvant par la même occasion les coûts de location associés. Le partage devra s'effectuer de manière transparente pour les clients du site web qui ne changeront nullement leur façon de s'y connecter, tout en offrant une bande passante suffisamment importante pour résister à la montée en charge. Pour ce faire, nous introduirons le concept de partageur, correspondant à un client du site qui, sachant que le site à une capacité d'être partageable, prendra la décision d'installer un logiciel sur son ordinateur. Il permettra ainsi d'absorber une partie de la charge totale du site lorsque son ordinateur sera allumé. Le logiciel en question utilisera la bande passante restante et non utilisée ainsi que l'espace disque que l'internaute a bien voulu lui concéder. Plus le nombre de partageurs est important, plus le site web partagé sera robuste autant en matière de disponibilité qu'en tentative de censure.

Nous passerons en revue le fonctionnement de plusieurs protocoles sur lesquels nous nous

appuierons par la suite et verrons également des projets se rapprochant de notre solution sans pour autant répondre à nos hypothèses de départ. Par la suite, nous détaillerons l'ensemble des serveurs et logiciels que nous utiliserons et la façon dont ils traiteront les requêtes des clients. L'objectif étant pour les clients de ne pas modifier leur comportement pour récupérer des pages web, nous expliquerons en détail le mécanisme que nous proposerons afin de répartir la charge du site web partagé, sans pour autant introduire de point central dans notre architecture. Nous utiliserons pour cela le protocole DNS, auquel nous couplerons le protocole Kademdlia fonctionnant sur le principe de tables de hachage distribuées. Pour cela, nous présenterons un mécanisme de conversion d'adresses IP en ASCII et inversement et utiliserons les différents types d'enregistrement DNS, comme les délégations de zones et les noms canoniques. Nous verrons ensuite comment nous permettons aux différents partageurs d'alléger le nombre de requêtes leur provenant en distribuant les ressources associées (images, scripts Javascript, feuilles de style...) qui seront demandées par les clients, une fois les pages HTML ou les feuilles CSS téléchargées. Pour récupérer initialement les ressources du site, le partageur devra les récupérer par pair-à-pair en utilisant le protocole Bittorent couplé avec Kademlia. Nous continuerons donc par expliquer comment le logiciel pair-à-pair sera calibré, quelles ressources il devra récupérer en priorité et comment il communique avec les autres programmes. Nous finirons notre prototypage par introduire des métriques et des seuils permettant la prise de décision pour les différents serveurs applicatifs.

Pour tester notre modèle, nous le comparerons à un une approche client/serveur avec un serveur en salle machine. Le débit, le nombre et la durée de connexion des partageurs dépendent de facteurs externes difficilement modélisables. Nous ferons donc des approximations en utilisant des études statistiques et comparerons les deux modèles sur les débits offerts dans un premier temps avant de nous pencher sur le taux de disponibilité. Les résultats de nos simulations nous montrerons des données encourageantes puisqu'avec un nombre de partageurs raisonnable (inférieur à 100) et un nombre important de clients, nous serons capable d'égaler les performances d'un serveur en salle machine, sur les critères que nous aurons préalablement défini.

ABSTRACT

The use of the Internet in our today's society has become an essential part of our daily life. Not only do we use it by accessing online government web services or commercial websites but also in order to communicate with others. This could be by email, instant messaging or by sharing articles on websites if the targeted audience is aimed to be as big as possible. Everyone has now the ability to share information reachable by anyone on the planet. The Internet is a network with no centre or head that may regulate it. As its name states, it is an inter-network, a collection of thousands of different networks interconnected between each other and using the same protocol called the Internet Protocol (IP). However, we can notice a growing asymmetry between the traffic of the different network operators these days. Some ISP would only focus on targeting end user customers while in the same time, the others would only offer hosting rental plans and have the vast majority of the content that users want. The consequences are a totally unbalanced inter-AS traffic, giving the possibility for new comers to interconnect very difficult, as well as raising concerns about the concentration of all the information by only several companies on privacy and data control issues. By the use of the today's Internet, artificial centres have been created. IP is a neutral protocol that doesn't treat differently packets coming from servers than the ones from clients. Its only job is to carry the packets trough the network to the recipient. Thus, server hosting on a domestic Internet access is possible but rather limited since the usual bandwidth is too low to support scalability.

Our project aims to tackle this issue by giving a simple and efficient way for Internet users to share contents without having to host those on a data-centre and in the same time being rid of monthly fees too. Web share will have to be transparent for the end users that will not change their way of sending requests. It also has to offer enough bandwidth to resist scalability. In order to do this, we are going to introduce the notion of sharer. A sharer is a client of the website that took the decision to install a piece of software on his/her computer, knowing that the website offers shared capabilities. All sharers, by doing this, will let clients of the website request contents on their computer when it's turned on. The piece of software will use the remaining bandwidth and the amount of space on the HDD/SSD the user would have let it first. The more the number of sharers is important, the more the shared website will be robust enough in term of availability as well as censorship.

We will first see all the detail mechanisms of several interesting protocols that we will use

on our solution such as HTTP, DNS as well as the different types of peer-to-peer protocols. We will also look over the different existing projects similar to what we aim to do but without answering to our initial requirements. Then, in the part of the establishment of the model, we will see which software we will use and how we answer client requests. The goal is for the client not to modify their usual behaviour on how they usualy proceed to retrieve web pages. We will then go through a detail explanation on how we manage load balancing between the different sharers without introducing a central point (or SPOF) in our architecture. We will use to answer this question, the DNS protocol that we will peer with the Kademlia protocol using DHT tables. To do this, we will introduce a way to translate IP addresses in ASCII characters and vice versa and we will take advantage of the different records offered by DNS such as zone delegation or canonical names. Then we will see how we manage to lighten the number of resource requests by distributing the associated related resources (such as images, Javascript scripts or CSS sheets) that would be requested by clients when downloads of HTML pages or CSS sheets have been completed. To initially retrieve the resources of the website , the sharers would have to download those on a peer-to-peer network using the Bittorent protocol peered with Kademlia. We will pursue by explaining how we calibrate the peer-to-peer application that we will use, which resources should be downloaded in priority and how it communicates whith the other pieces of software we will be using. Finally, we will introduce metrics and thresholds letting the different servers to take a decision on how they should behave.

To evaluate our model, we will compare it to a client/server based server stored in a data-centre. The throughput, the number of sharers and the duration of the connections are external factors difficult to emulate since they are based on human interaction, especially the duration evaluation. We will make approximations and add random factors in our statistic studies in order to make the comparison bewteen our two models on throughput first before going trough disponibily. Our results will show interesting data. With a rather limited number of sharers (under 100) we will be able to reach the same results and quality compared to a server in a data-centre on the critera we will have defined before.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xii

LISTE DES FIGURES xiii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

CHAPITRE 1 INTRODUCTION 1
1.1 Définitions et concepts de base 1
1.2 Éléments de la problématique 2
1.3 Objectifs de recherche 3
1.4 Plan du mémoire 4

CHAPITRE 2 CARACTÉRISATION DES RÉSEAUX TCP/IP ET DES APPROCHES PAIR-À-PAIR 5
2.1 Le protocole DNS 5
2.1.1 Les enregistrements DNS 6
2.1.2 *Time to live* 8
2.1.3 Le fonctionnement sur Internet 8
2.2 Le protocole HTTP 9
2.2.1 HTTP/0.9 9
2.2.2 HTTP/1.0 et HTTP/1.1 10
2.3 Topologie des réseaux 12
2.3.1 Topologie en étoile 12
2.3.2 Topologie maillée 13
2.3.3 Topologie du réseau Internet 15

2.4 Les modes de communications sur Internet 16
2.4.1 L'environnement client/serveur 16
2.4.2 L'environnement pair-à-pair 17
2.4.3 Comparaison des deux modèles 17
2.5 Les réseaux pair-à-pair 18
2.5.1 L'évolution des réseaux pair-à-pair 18
2.5.2 Réseaux pair-à-pair structurés et réseaux non-structurés 20
2.5.3 Les tables de hachage distribuées 21
2.6 Kademlia 22
2.6.1 L'adressage Kademlia 22
2.6.2 La métrique de distance 22
2.6.3 La connaissance du réseau 23
2.6.4 La recherche d'un nœud ou d'une ressource 24
2.6.5 Les instructions Kademlia 25
2.6.6 L'introduction dans le réseau 26
2.7 Projets similaires 27
2.7.1 Projet DIASPORA* et Statusnet 27
2.7.2 Projet Bittcoin et NameCoin 27
2.7.3 Projet P2PWeb 28
2.7.4 Projet WebRTC 29

CHAPITRE 3 STRATÉGIE DE DÉCENTRALISATION PROPOSÉE 30
3.1 Définitions 30
3.2 Hypothèses 31
3.3 Description sommaire de la solution 32
3.4 Différents acteurs du réseau 33
3.4.1 Client 33
3.4.2 Partageur 33
3.5 Préférences du partageur 34
3.6 Stratégie de répartition de la bande passante 35
3.7 Le serveur DNS autorité 35
3.7.1 Délégation d'une requête à un autre partageur 36
3.7.2 Traitement d'une résolution de type www.example.com 38
3.7.3 Gestion des TTL 38
3.7.4 Résumé de la gestion DNS 39
3.8 Le serveur web HTTP 40

3.8.1 Enregistrement de la topologie du site 40
3.8.2 Gestion des requêtes des clients 42
3.8.3 Envoi des ressources aux clients 43
3.9 Servent pair-à-pair 45
3.10 Gestion de la bande passante 46
3.10.1 Répartition et réservation de débit 46
3.10.2 Priorisation 48
3.10.3 Répartition de récupération des ressources 49
3.10.4 Indexation préventive 49
3.11 Gestion de l'espace disque 50
3.12 Gestion des DNS dans Internet 51
3.12.1 Processus d'élection 51

CHAPITRE 4 ANALYSE DE PERFORMANCE 55
4.1 Définition des critères de performance 55
4.2 Valeurs de nos modèles 55
4.3 Débit constaté 56
4.4 Disponibilité 64

CHAPITRE 5 CONCLUSION 71
5.1 Synthèse des travaux 71
5.2 Limitations de la solution proposée 72
5.3 Améliorations futures 72
5.3.1 Sites dynamiques 72
5.3.2 Mode hybride 72
5.3.3 Limitation du volume de transferts 73
5.3.4 Sécurité 73
5.3.5 Rapprocher les clients et partageurs 73

RÉFÉRENCES 74

LISTE DES TABLEAUX

TABLEAU 3.1 Comportement du DNS d'un partageur non surchargé (cas 1) 39
TABLEAU 3.2 Comportement du DNS d'un partageur surchargé (cas 1) 39
TABLEAU 3.3 Comportement du DNS d'un partageur non surchargé (cas 2) 40
TABLEAU 3.4 Comportement du DNS d'un partageur surchargé (cas 2) 40
TABLEAU 3.5 Caractéristiques des sommets du graphe d'un site partagé 41
TABLEAU 3.6 Caractéristiques des zones injectables 44
TABLEAU 4.1 Débits maximums 56
TABLEAU 4.2 Débits maximums 57
TABLEAU 4.3 Page de l'IETF et ses dépendances 57
TABLEAU 4.4 Volume de données parallélisables pour la page d'accueil de l'IETF (en ko) 59
TABLEAU 4.5 Débits observés sur le réseau en fonction du degré de parallélisation pour la page d'accueil de l'IETF (en kbps) 59
TABLEAU 4.6 Débit observé pour un client donné en fonction du degré de parallélisation pour la page d'accueil de l'IETF (en kbps) 60
TABLEAU 4.7 Capacité maximale théorique du réseau de partageurs (kbps) 60
TABLEAU 4.8 Résultats de l'execution d'une heuristique taboue pour le site de l'IETF 64

LISTE DES FIGURES

FIGURE 2.1 Fonctionnement d'une résolution récursive 6
FIGURE 2.2 Deux types de topologie : en étoile et maillée. 12
FIGURE 2.3 Exemple de topologie d'interconnexions d'opérateurs Internet. 16
FIGURE 2.4 Exemple de tables de hachage. 21
FIGURE 2.5 Exemple de réseau Kademlia et de stockage *k-bucket*. 24
FIGURE 3.1 Approche sommaire de la solution 32
FIGURE 3.2 Diagramme d'activité du comportement DNS de notre modèle avec un partageur ayant l'adresse 2001 :db8 : :1 53
FIGURE 3.3 Diagramme d'activité du comportement HTTP de notre modèle . . . 54
FIGURE 4.1 Volume de données parallélisables (en ko) 61
FIGURE 4.2 Débit total utilisable par le réseau de partageurs (en kbps) 61
FIGURE 4.3 Rapport débit utilisable / débit disponible (en pourcentage) 62
FIGURE 4.4 Débit de transfert effectif pour chaque client (en kbps) 62
FIGURE 4.5 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,5% 67
FIGURE 4.6 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,999% 68
FIGURE 4.7 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,999% (autre échelle) 69

LISTE DES SIGLES ET ABRÉVIATIONS

ACEI	Autorité Canadienne pour les Enregistrements Internet
API	Application programming interface
ARIN	American Registry for Internet Numbers
ASCII	American Standard Code for Information Interchange
ASN	Autonomous System Number
BGP	Border Gateway Protocol
BRAS	Broadband Remote Access Server
ccTLD	country code Top-Level Domain
CIRA	Canadian Internet Registration Authority
CSS	Cascading Style Sheets
CDN	Content Delivery Network
DHT	Distributed Hash Table
DNS	Domain Name System
DSL	Digital Subscriber Line
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
gTLD	generic Top-Level Domain
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
JS	JavaScript
LIR	Local Internet Registry
MIME	Multipurpose Internet Mail Extensions
MX	Mail eXchanger
NAT	Network Address Translation
OSI	Open Systems Interconnection

OSPF	Open Shortest Path First
POP	Post Office Protocol
QoS	Quality of Service
RFC	Request For Comments
RIR	Regional Internet Registry
SLAAC	Stateless Address AutoConfiguration
SMTP	Simple Mail Transfer Protocol
SPOF	Single Point Of Failure
SQL	Structured Query Language
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLD	Top-Level Domain
TTL	Time To Live
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

CHAPITRE 1

INTRODUCTION

Le réseau Internet a depuis une quinzaine d'années bouleversé la façon dont les humains communiquent et échangent l'information entre eux abolissant ainsi les distances et les canaux classiques d'information. Le succès de ce réseau d'interconnexions est dû en particulier aux standards publics, ouverts et gratuits permettant à n'importe qui de déployer un réseau TCP/IP n'importe où dans le monde. Ce réseau, a contrario des anciens réseaux de l'époque avait à la fois la particularité de ne disposer structurellement d'aucun centre et de développer à grand échelle la communication sous forme de datagrammes en introduisant la notion de paquets. Aujourd'hui, la majorité de l'utilisation qui est faite du réseau utilise le protocole HTTP pour transmettre des pages web et nous assistons même à une convergence des autres protocoles vers HTTP.

Dans ce chapitre, nous énoncerons les concepts de base du réseau Internet et de la suite TCP/IP ainsi que les deux grands modèles de réseaux informatiques que l'on y retrouve. Par la suite, nous expliquerons les éléments de la problématique sur lesquelles nous nous pencherons avant d'énoncer les objectifs de ce mémoire et les résultats que nous souhaiterions obtenir. Nous terminerons par présenter la structure du mémoire.

1.1 Définitions et concepts de base

Un réseau TCP/IP est un réseau utilisant une suite de protocoles normalisés afin de permettre la communication entre tous les appareils du réseau. Chaque appareil connecté à Internet dispose d'une adresse IPv4 ou IPv6 qui lui est propre et qui est unique mondialement. N'importe quel appareil connecté à Internet et disposant d'une adresse peut envoyer n'importe quel message à n'importe quel autre appareil connecté en utilisant le protocole IP et en spécifiant l'adresse IP du destinataire dans le champ consacré. Il sera également possible pour ce dernier d'obtenir une réponse de la même manière, en changeant les champs des adresses du destinataire et de l'expéditeur. Le protocole IP ne fait pas de distinction entre une question et une réponse et les routeurs, chargés d'acheminer les paquets IP à leur destinataire, ne font qu'office d'intermédiaires. Au dessus du protocole IP fonctionne un grand nombre de protocoles dont les protocoles applicatifs permettant différents usages comme la

consultation ou l'envoi de courriels, l'échange de fichier ou bien l'affichage de pages disposant de lien hypertextes et capables d'afficher des images et une mise en page.

Parmi les protocoles applicatifs, on retrouve deux grandes familles, à savoir les modèles clients/serveurs et les modèles pair-à-pair qui ont une approche différente de la communication entre les différents appareils connectés au réseau :

— Le premier considère un modèle $* \dots 1$ avec des appareils étant les clients demandant des ressources à un appareil unique, appelé serveur, satisfaisant l'ensemble des demandes. Ce type de modèle est le modèle le plus simple et le plus intuitif. Il possède néanmoins quelques désavantages comme nous le verrons plus tard, en particulier celui de résister relativement mal à l'évolutivité. En effet, en faisant grandir le nombre de clients, notre serveur n'aura plus les capacités matériels à répondre à toutes les demandes.

— Le second considère, quant à lui, un modèle $* \cdots *$ ou tous les appareils faisant partie de ce réseau récupèrent des ressources auprès de n'importe lesquels des autres appareils du réseau et envoyant également de son côté des ressources à d'autres membres du réseau. Ce type de modèle est bien plus complexe à modéliser, en particulier si nous souhaitons nous affranchir de tout point central. Il a fait, et fait toujours, l'objet de beaucoup de travaux de recherche. Contrairement au premier modèle, ce modèle-ci résiste très bien aux demandes importantes de nouveaux appareils, puisque chaque nouvel appareil, en plus de venir récupérer des ressources, met à disposition de nouvelles capacités pour faire croître le réseau.

L'ensemble des protocoles réseaux fonctionne par un système de couches où chaque couche s'appuie sur les couches inférieures et fournit des services aux couches supérieures sans en connaître le fonctionnement interne. Ainsi, le protocole de routage IP ne fait aucune distinction quant aux protocoles utilisés par les couches supérieures et de nouveaux protocoles applicatifs peuvent ainsi être crées et être directement utilisables sur Internet sans opérer aucun changement quant aux infrastructures.

1.2 Éléments de la problématique

Les principaux défis dans les réseaux aujourd'hui sont multiples. Tout d'abord, d'un point de vue de la consommation énergétique, les appareils connectés représentent une masse im-

portante de la consommation énergétique mondiale. Cela est particulièrement le cas dans les fermes de serveurs gigantesques de certains opérateurs réseaux spécialisés dans l'hébergement de contenu. D'un point de vue de la société, le réseau Internet est devenu l'outil essentiel d'échanges de communication et le facteur de liberté d'expression à travers le monde ; il est donc tout naturellement sujet à censure. À cela s'ajoute le désir grandissant pour les utilisateurs d'Internet de contrôler leur vie privée et leurs contenus numériques. Du côté des opérateurs réseaux, ces derniers se sont pour la plupart spécialisés soit dans une fourniture d'accès à Internet pour les particuliers ne proposant qu'un faible débit montant *a contrario* du débit descendant alors que d'autres ne se proposent que d'héberger des services en ligne. Cela a donc pour effet une asymétrie importante entre le trafic des différents opérateurs réseaux et la création de points centraux névralgiques sur le réseau qui sont néfastes tant d'un point de vue technologique, en multipliant les risques de défaillances, qu'administratif, en ne confiant toute l'information qu'à quelques compagnies ayant le contrôle dessus.

Ce mémoire vise à proposer un modèle prenant pleinement avantage du réseau Internet pour les applications web utilisant le protocole HTTP en y appliquant le modèle du réseau pair-à-pair tel qu'on le retrouve aujourd'hui sur les réseaux de partage de fichiers. Notre modèle visera donc à permettre à tous les nœuds du réseau pair-à-pair du site web d'agir en tant que serveurs et de répondre aux requêtes des clients, ces derniers ne devant pas changer la façon dont ils récupéreront les ressources. Nous veillerons également à n'introduire aucun centre dans notre modèle. Cela soulève donc des problèmes auxquels nous devrons répondre. La première question que nous devrons nous poser et de savoir comment ces nœuds du réseaux pourront répondre de manière transparente aux clients et faire la liaison avec le réseau pair-à-pair. La seconde, encore plus importante, visera à trouver un mécanisme permettant à un client de chercher de manière dynamique dans le réseau pair-à-pair des serveurs/nœuds pouvant satisfaire ses demandes mais sans changer la façon dont il se comporte pour récupérer l'information. Une autre difficulté à laquelle nous serons aussi confrontés sera la gestion du débit des nœuds du réseau pair-à-pair qui peut rapidement arriver à saturation et comment nous pouvons permettre aux clients de ne pas être trop ralenti lors de telles congestions en répartissant la charge sur plusieurs nœuds, la gestion du débit étant primordiale dans notre approche.

1.3 Objectifs de recherche

L'objectif principal de ce mémoire est de concevoir un modèle permettant de distribuer en pair-à-pair des sites web entre différentes machines et dans le même temps permettre aux

clients du site d'y accéder en mode client/serveur tel qu'ils le font traditionnellement. Le système proposé fera donc la liaison entre l'approche pair-à-pair et l'approche client/serveur de manière transparente. Plus particulièrement, nous nous proposons de :

— Analyser les différents protocoles que nous utiliserons afin de choisir les plus pertinents et mettre en valeur les paramètres sur lesquels nous pourrions agir par la suite. Analyser également les différents travaux de recherche et solutions déjà existantes et se rapprochant de notre objectif.

— Concevoir un modèle du fonctionnement des partageurs, machine étant à la fois nœud du réseau pair-à-pair et serveur des clients du site permettant de :
  — Décentraliser autant que possible le site web partagé en permettant une recherche du meilleur serveur par le client de manière transparente sans introduire de point central dans le réseau.
  — Répartir autant que possible la charge du site sur le plus de partageurs afin de permettre de meilleures performances.
  — Gérer les ressources allouées par les partageurs et en tirer le meilleur profit.
  — Développer des algorithmes de prise de décision pour les partageurs.
  — Gérer l'intégration de notre modèle dans l'implémentation hiérarchique du DNS tel qu'on le trouve sur Internet (ICANN).

— Évaluer et analyser la performance de notre modèle en introduisant des mesures et en les testant autant analytiquement qu'en simulation.

1.4 Plan du mémoire

Ce mémoire est composé de cinq chapitres. Après ce présent chapitre d'introduction, nous allons faire un état de l'art sélectif des différentes technologies et recherches ayant trait à notre sujet et qui nous servirons pour le reste du mémoire. Le troisième chapitre proposera une solution à notre problématique en décrivant son fonctionnement dans les détails. Le quatrième chapitre aura pour objectif de présenter les mesures de performances qui ont été effectuées afin de tester l'efficacité de notre modèle. Enfin, nous conclurons par un dernier chapitre en faisant une synthèse du travail accompli et en proposant des axes d'amélioration pour de travaux futurs de recherche.

CHAPITRE 2

CARACTÉRISATION DES RÉSEAUX TCP/IP ET DES APPROCHES PAIR-À-PAIR

Un réseau TCP/IP (IP pour *Internet protocol*) est un réseau informatique utilisant la suite de protocoles TCP/IP sur laquelle Internet s'appuie pour fonctionner. TCP/IP est une adaptation simplifiée du modèle OSI d'ISO. La plupart des réseaux modernes utilisent aujourd'hui TCP/IP, que cela soit sur des réseaux publics comme Internet ou bien dans des réseaux internes privés. Un réseau pair-à-pair (ou P2P) est un réseau décentralisé qui se base sur la capacité des ressources de tous ses participants. Chaque participant est à la fois consommateur et diffuseur. Ce type de réseau s'oppose traditionnellement aux réseaux clients/serveurs où toutes les ressources sont détenues par les serveurs qui les mettent à la disposition des clients. Les réseaux centralisés posent des problèmes inhérents à leur structure lorsqu'on l'on observe un accroissement substantiel de la demande, ce que l'on ne retrouve pas dans des approches décentralisées. Dans le présent chapitre, nous nous proposons dans un premier temps de passer en revue quelques protocoles populaires utilisés dans les réseaux IP en mettant l'emphase sur des caractéristiques qui nous semblent importantes pour par la suite brosser un état de l'art des réseaux pair-à-pair et en particulier ceux entièrement distribués. Nous passerons enfin en revue quelques projets ayant des similarités avec le nôtre.

2.1 Le protocole DNS

Le protocole DNS (*Domain Name System*) est un des protocoles les plus importants, si ce n'est le plus important, de l'Internet moderne fonctionnant en TCP et UDP sur le port 53. Bien qu'il soit un protocole applicatif (couche 7), il agit étroitement avec la couche réseau en particulier pour la résolution de noms. À l'image d'un annuaire téléphonique, il permet principalement de convertir un nom en un numéro. L'usage le plus courant sur Internet consiste à résoudre un nom de domaine comme `www.example.com` en une adresse IP comme `172.16.1.2` ou en `2001:db8::1` afin de pouvoir contacter la machine possédant cette adresse pour lui demander, par exemple, la page principale d'un site web qu'il héberge.

Alors qu'il aurait été tout à fait possible d'adopter une architecture centralisée en mettant l'annuaire mondial de résolution de noms sur une seule machine servant de référence pour toutes les résolutions, les concepteurs de ce protocole ont pris dès le début la décision de

décentraliser au maximum cet annuaire (référence vers la RFC originelle). L'idée principale est d'avoir la possibilité de déléguer à un autre serveur, appelé serveur autorité, des zones permettant la création de sous-domaines. Un nom de domaine se lit de droite à gauche et est segmenté par le séparateur point (".") à partir duquel on peut choisir de déléguer le reste de la résolution à une autre autorité. L'architecture globale ressemble ainsi à un arbre à branches multiples avec à son sommet les serveurs racine.

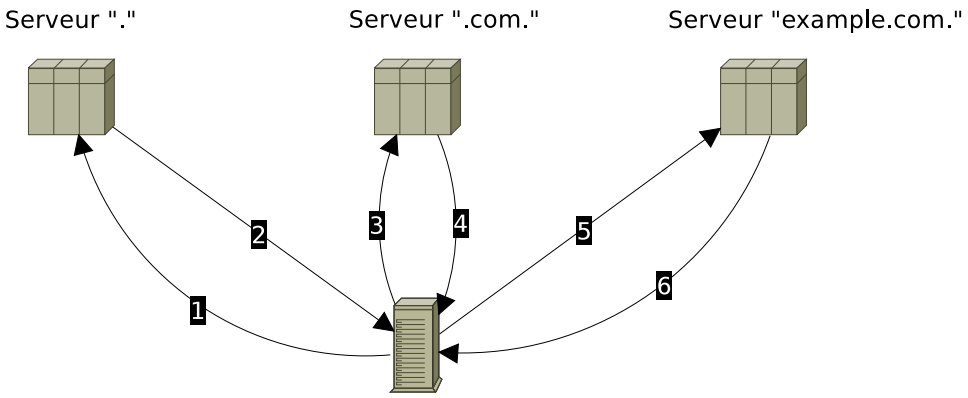

FIGURE 2.1 Fonctionnement d'une résolution récursive

Par exemple dans la figure 2.1, un serveur résolveur ou cache souhaite trouver l'adresse IP correspondant au domaine `www.example.com`. Pour ce faire il va contacter de manière récursive plusieurs serveurs afin d'avoir une réponse. Ne connaissant que l'adresse du serveur racine dans un premier temps, ce qui est un pré-requis, il demande à ce dernier de résoudre `www.example.com` (requête **1**). Ce serveur ne connaît pas la réponse à cette requête mais connaît le serveur faisant autorité sur `.com` dont le domaine demandé en est un sous-domaine. La réponse (**2**) correspond donc à l'adresse de ce nouveau serveur qui est contacté dans la foulée. Une requête similaire à **1** est faite (**3**) et la réponse (**4**) renvoie le demandeur sur un autre serveur qui est responsable de `example.com`. Pour finir, la requête **5** (toujours identique à **1** et **3**) aura pour réponse la résolution demandée. Cet exemple permet d'illustrer l'aspect récursif du protocole et l'approche choisie pour le décentraliser. Il est important de noter que les deux premiers serveurs auraient très bien pu répondre directement à la requête sans la déléguer à un autre serveur.

2.1.1 Les enregistrements DNS

Les requêtes et les réponses des serveurs DNS ne se résument pas seulement à des résolutions de noms de domaine pleinement qualifiés (FQDN) en adresses IP. Il existe en

réalité plusieurs types de messages appelés enregistrements (*records*). Nous nous proposons ici d'en lister quelques uns qui seront pertinents pour la suite de notre étude :

Enregistrements A et AAAA Ce type d'enregistrement est une demande de résolution d'un nom de domaine par une adresse IPv4 s'il s'agit d'un enregistrement A ou IPv6 pour les enregistrements AAAA et est envoyée par le résolveur. Le serveur peut soit répondre par une réponse A ou AAAA comme dans **6**, ou bien par un CNAME ou un NS comme décrit ci-après.

Enregistrements NS Ce type d'enregistrement est une réponse d'un serveur faisant autorité sur une zone. Le serveur par cette réponse délègue une sous-zone à un autre serveur dont l'adresse est précisée avec la réponse. Les réponses **2** et **4** de la figure 2.1 sont des réponses de type NS.

Enregistrements SOA Ce type d'enregistrement correspond à une requête sur un nom de domaine. Le serveur faisant autorité sur la zone chargée de la résolution du domaine demandé répondra par un SOA avec des informations sur la zone.

Enregistrements CNAME et DNAME Ces enregistrements sont la plupart du temps des réponses à des requêtes A ou AAAA. Au lieu de répondre avec un enregistrement similaire, le serveur répond par un *Canonical Name*, abrégé en CNAME ou un DNAME indiquant que le domaine demandé n'est qu'un alias d'un autre domaine précisé en réponse. Si la réponse est un CNAME, la résolution s'arrête ici et le résolveur DNS peut éventuellement faire une nouvelle résolution avec le nouveau domaine demandé. S'il s'agit d'un DNAME la RFC 1035 (Mockapetris (1987)) précise que la résolution du nouveau domaine doit se faire dans la foulée.

Enregistrements PTR Ce type de requête correspond à l'inverse des requêtes A/AAAA. Les requêtes de type PTR demandent le nom de domaine qui est associé à une adresse IP donnée. L'adresse IP est transcrite en un nom de domaine d'un sous-domaine de la zone `.in-addr.arpa` pour les adresses IPv4 et `.ip6.arpa` pour les adresses IPv6.

Enregistrements MX Ce type d'enregistrement correspond à une demande du nom du serveur s'occupant de réceptionner les courriels pour le domaine demandé.

Enregistrement TXT Ces enregistrements permettent de retourner des chaînes de caractères pour un FQDN donné.

2.1.2 *Time to live*

Toutes les réponses DNS, quelque soit le type d'enregistrement, se voient associer un champ TTL (*Time to live*) correspondant à la durée pendant laquelle la réponse est réputée valide. Ainsi, si le résolveur souhaite résoudre de nouveau la même requête, il regardera dans son cache avant de procéder réellement à l'émission de la requête et utilisera la réponse enregistrée si sa durée de vie est encore valide. Il est malheureusement à noter que selon l'implémentation du serveur récursif, ce champ est plus ou moins bien interprété.

2.1.3 Le fonctionnement sur Internet

Nous venons de voir rapidement le fonctionnement technique du protocole DNS et en particulier la notion de délégation de zone. Cela implique nécessairement de connaître l'adresse du serveur racine *a priori* sans quoi aucune résolution n'est possible. Dans l'objectif que tout le monde puisse résoudre sans conflit le même nom de domaine par la même adresse, la ICANN (*Internet Corporation for Assigned Names and Numbers*) par le biais de l'IANA (*Internet Assigned Numbers Authority*) a décidé de déléguer à treize entités administratrices la charge d'héberger des serveurs racine. Ces serveurs sont redondants et répartis sur toute la surface de la planète pour la plupart en utilisant un mécanisme d'adressage *anycast* pour le routage. Il est important de bien noter que l'utilisation de ces serveurs se fait par convention mais n'est aucunement une règle absolue. Il existe ainsi des serveurs racine dits alternatifs comme ORSN ou 42registry ou bien encore des projets de décentralisation en pair-à-pair de la racine dont nous parlerons plus loin. L'IAB (*Internet Architecture Board*) s'est cependant positionné contre ce genre de pratique (Board (2000)) tout comme la ICANN elle-même (notice ICP-3).

Les serveurs racines ne délèguent pas des zones de premier niveau directement à des particuliers ou à des entreprises mais à des organismes appelés registres (*registries*) chargés de déléguer des sous-zones selon leur politique de délégation. Les registres peuvent être des organismes étatiques ou des entreprises ou associations ayant un objectif de service public, principalement pour les noms régionaux aussi appelés ccTLD (*Country code top-level domains*). C'est le cas par exemple de l'ACEI au Canada qui a en charge la zone `.ca` et qui délègue des sous-zones de second niveau (`.ca`) et qui déléguait aussi des sous-zones de troisième niveau comme `.qc.ca` par exemple. Par opposition aux ccTLD, on retrouve les gTLD (*Global top-level domains*) qui fonctionnent techniquement et administrativement de la même manière mais qui sont associés, tout du moins à l'origine, à des types d'activités comme les `.com` pour les serveurs a visées commerciales ou encore les `.net` pour les serveurs

en rapport avec le fonctionnement technique du réseau.

Les registres les plus populaires (comme les `.com`, `.net`, `.org` ou les ccTLD nationaux) qui croulent sous les demandes de délégations, ont pris la décision de déléguer la relation clientèle et technique en partie à des entreprises externes appelées registraires ou bureaux d'enregistrement (*registars*). Le particulier ou l'entreprise souhaitant se voir déléguer une sous-zone doit obligatoirement s'adresser à un registraire homologué par le registre en question en lui précisant toutes les informations nécessaires à l'enregistrement. La plupart des registraires en profitent pour vendre des services supplémentaires, comme l'hébergement de serveurs faisant autorités sur la zone louée ou même d'autres services comme l'hébergement de pages web ou de comptes courriel.

2.2 Le protocole HTTP

Le protocole HTTP est le protocole développé pour le *World Wide Web*. Fonctionnant en mode client/serveur, il s'agit d'un protocole applicatif fonctionnant sur TCP et utilisant par défaut les ports 80 et 443 pour sa version sécurisée (HTTPS, HTTP sur TLS) selon les *Well know ports* de l'IANA. Il est aujourd'hui le protocole le plus populaire de l'Internet et le plus connu de la majorité des personnes à tel point que le web est souvent confondu avec l'Internet.

Le Web est un système permettant l'accès à des ressources connectées entre elles par le bais de liens hypertextes. Le protocole HTTP est le protocole permettant l'accès par le biais d'un serveur web côté serveur, et le téléchargement et l'affichage de ces ressources grâce à un navigateur ou fureteur web côté client. Les ressources web consultables et affichables sur un navigateur sont des fichiers au format (X)HTML respectant les standards du W3C (*World Wide Web Consortium*).

L'objectif de la présente section est de présenter succinctement le protocole HTTP, son fonctionnement et les options dont nous aurons besoin par la suite. Le Web en lui même et les langages associés ne sont que peu importants pour notre étude, il est cependant nécessaire de garder à l'esprit la notion d'hyperlien que nous utiliserons.

2.2.1 HTTP/0.9

À ses tout débuts et avant d'être normalisé par une RFC, Tim Berners-Lee du MIT propose ce protocole ainsi que le principe du Web. Le fonctionnement de la version 0.9 est extrêmement simple. Après l'établissement d'une connexion au serveur web, le client précise la

ressource qu'il souhaite récupérer après le mot clef `GET`. Une fois la requête effectuée et la ressource envoyée, la connexion TCP est terminée. L'exemple suivant illustre le téléchargement de la page d'accueil d'un site :

```
GET /index.html
```

Le serveur nous répondra par la suite avec un code de retour `HTTP 200 OK` si la requête a été exécutée avec succès et suivi du contenu de la page demandée.

2.2.2 HTTP/1.0 et HTTP/1.1

Très rapidement après HTTP/0.9, le HTTP Working Group a décidé d'améliorer cette version et de créer la version 1.0 puis 1.1 (Fielding *et al.* (1999)) en ajoutant de nombreuses nouvelles fonctionnalités. La version la plus à jour est la version 1.1 et est utilisée en très grande majorité sur Internet aujourd'hui. En plus de permettre des connexions persistantes, les versions 1.0 et 1.1 permettent d'ajouter des en-têtes lors de la formulation de requêtes HTTP au serveur. Nous allons par la suite détailler quelques une de fonctionnalités qui nous semblent utiles pour la suite de notre projet :

Les hôtes virtuels

Une des avancées majeures d'HTTP/1.0 et HTTP/1.1 a été la notion d'hôtes virtuels (*virtual hosts*) permettant d'associer plusieurs sites à une même adresse IP. Avec HTTP/0.9, il n'était pas possible de préciser le site web demandé, le client ne précisant pas le FQDN du site web qu'il souhaitait joindre. Par conséquent, si plusieurs FQDN résolvaient la même adresse IP, le site web retourné par le serveur aurait été le même. HTTP/1.1 exige dorénavant l'ajout de l'en-tête `Host:` suivi du FQDN avec lequel le client souhaite se connecter (la version 1.0 l'introduisait de manière optionnelle). Ainsi, dépendemment du FQDN, le serveur ne retournera pas la même page, même si la ressource demandée est la même. Par exemple, si `www.example.com` et `www2.example.com` sont résolus par la même adresse `192.168.1.1`, ces deux requêtes ne donneront pas nécessairement le même résultat :

```
GET /index.html HTTP/1.1
Host: www.example.com
```

```
GET /index.html HTTP/1.1
Host: www2.example.com
```

Téléchargement partiel

La version 0.9 et 1.0 du protocole ne permettait pas de ne télécharger qu'une partie d'une ressource. Par conséquent, si un téléchargement était arrêté avant la fin, il n'y avait pas d'autre moyen que de redemander la même ressource en récupérant de nouveau les premiers

octets qui avaient déjà été téléchargés. Le protocole HTTP/1.1 pâlie ce problème en proposant l'option `Range:` permettant de préciser l'intervalle que l'on souhaite récupérer. Cela est donc particulièrement utile pour la récupération de gros fichiers. Bien que HTTP/1.1 ajoute cette nouvelle fonctionnalité, c'est au serveur web de décider s'il accepte des requêtes `Range:` ou non. De manière générale le serveur l'indique dans ses en-têtes de réponse, si tel est le cas, avec une en-tête par exemple comme `Accept-Ranges: bytes` en précisant aussi l'unité de subdivision. Ainsi, pour récupérer 100 Mo se trouvant après les 200 premiers Mo, on fera la première requête, alors que si l'on souhaite récupérer tous les octets après les 500 premiers Mo, on fera la deuxième :

```
GET /video.avi HTTP/1.1
Host: www.example.com
Range: bytes=209715200-314572800
```

```
GET /video.avi HTTP/1.1
Host: www.example.com
Range: bytes=524288000-
```

Compression des messages

Les versions 1.0 et 1.1 d'HTTP apportent également la possibilité de compresser à la volée les requêtes et les réponses HTTP, si le client et le serveur sont capables de les compresser et de les décompresser. Ainsi, lors d'une requête ou d'une réponse, le client ou le serveur peuvent préciser qu'ils acceptent qu'on converse avec eux en compressant les messages avec les formats de compression précisés. Pour ce faire, l'en-tête `Accept-Encoding:` a été proposée et permet de préciser les formats. Il est important de bien comprendre cependant qu'il s'agit juste d'une information et que le serveur ou le client ont tout à fait la possibilité d'ignorer cette en-tête en envoyant une réponse non compressée. Des formats de compression populaires sont gzip ou bzip2.

Les connexions persistantes

Pour terminer sur HTTP, la version 1.1 apporte une avancée majeure en terme d'optimisation, à savoir les connexions persistantes. Avec la version 0.9 et 1.0, le serveur refermait la connexion une fois la requête honorée et par conséquent si le client souhaitait récupérer une autre ressource sur le même serveur, il devait initier un nouveau *socket*, initiant une nouvelle synchronisation TCP et donc une perte de temps et de ressources. Il est cependant très courant d'avoir une page HTML faisant référence à un fichier CSS et à plusieurs images stockées sur le même serveur, entraînant ainsi plusieurs requêtes en cascades. La version HTTP/1.1 apporte une nouvelle option `Connection: Keep-Alive` qui permet de garder la connexion ouverte et d'enchaîner plusieurs requêtes à la suite.

2.3 Topologie des réseaux

Nous allons à présent prendre un peu de recul et nous intéresser aux différents types de réseaux qui existent, dans un premier temps d'un point de vue topologique, puis de manière applicative. Il existe un peu moins de dix grands types de topologie de réseaux mais nous allons n'en détailler ici que deux qui sont les plus pertinents pour la suite et très représentatifs de la majorité des réseaux d'aujourd'hui, c'est-à-dire les architectures en forme d'étoile et celles qui sont maillées comme représenté par la figure 2.2.

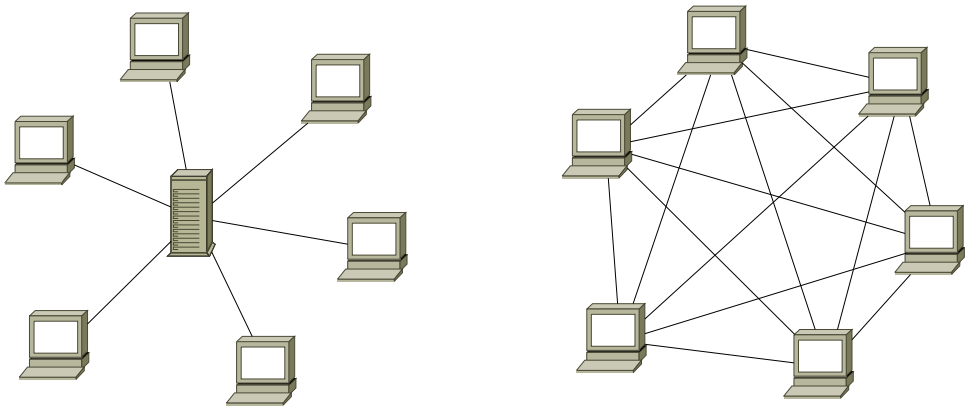

FIGURE 2.2 Deux types de topologie : en étoile et maillée.

2.3.1 Topologie en étoile

La topologie en forme d'étoile est, comme son nom l'indique, une topologie avec un nœud central chargé de permettre la communication de tous les nœuds se trouvant en périphérie. Ce nœud central peut également faire office de serveur central ou tous les terminaux viennent se connecter pour récupérer l'information. Ce type de réseau correspond historiquement à la topologie des premiers grands réseaux déployés. C'est par exemple l'architecture de tous les *vidéotexts* des années 1980 et début 1990, tel le Telidon développé par le CRC Canada ou le Minitel par France Télécom.

Avantages de la topologie en étoile

— La centralisation : Il est beaucoup plus facile pour gérer et contrôler le réseau d'avoir un point central par lequel passe tous les échanges sur le réseau. L'ajout de nouveaux nœuds au réseau est plus facile puisqu'une seule liaison est nécessaire. Il est de même plus facile de protéger le réseau de comportements contraire à sa politique en

le déconnectant éventuellement.

— De meilleures performances : En supposant que les capacités du nœud central augmente avec le nombre de nouveaux clients raccordés, les performances notamment en matière de temps de latence et de gigue seront nécessairement meilleures que dans un réseau totalement maillé puisque tous les nœuds sont joignables par l'ensemble du réseau en deux sauts. De même il n'est pas nécessaire d'utiliser des protocoles de routage dynamique (comme OSPF ou RIP dans IP), toute l'information, et de manière générale "toute l'intelligence" du réseau, étant détenu au niveau du nœud central.

Désavantages de la topologie en étoile

— Un point unique de défaillance : Le plus gros problème de cette topologie est son point unique de défaillance (*Single Point of Failure* ou *SPOF*) au niveau du nœud central. Si ce dernier était amené à ne plus fonctionner, le réseau entier ne serait plus opérationnel.

— Des problèmes d'extensibilité (*scalability*) : Ce type de réseau évolue très mal lorsque le réseau devient de plus en plus populaire et que de plus en plus de machines se raccordent. Pour palier ce problème, la solution classique consiste à augmenter les capacités du nœud central en même temps que le réseau. Cette solution trouve cependant ses limites dans la limite maximale des capacités possibles allouables au nœud central. Un réseau aussi populaire que ne l'est Internet n'aurait pas la possibilité physique de fonctionner de la sorte.

— La centralisation : Bien qu'elle fasse aussi partie des avantages de cette topologie, la centralisation peut également être vue comme quelque chose de néfaste pour le réseau, en plus des raisons citées précédemment. L'ensemble du réseau repose sur le nœud central et par conséquent le contrôle et le pouvoir de ce réseau dépendent exclusivement de l'autorité ayant le contrôle de ce nœud. Si cette autorité a donc la possibilité d'écouter, de modifier voire de censurer l'ensemble des communications circulant sur le réseau. De même, elle est la seule autorité possible à pouvoir faire évoluer ou non le réseau techniquement.

2.3.2 Topologie maillée

À l'opposé de cette topologie se trouve la topologie maillée voire totalement maillée (*fully meshed*). Dans cette topologie, chaque nœud du réseau est relié directement à tous les autres

nœuds du réseau. Dans la suite nous considérerons une topologie totalement maillée bien que dans les réseaux déployés à grande échelle, cela n'est pas le cas.

Avantages de la topologie maillée

— Pas de point unique de défaillance : L'un des avantages majeurs de ce type de topologie est l'absence totale de centre et les conséquences immédiates qui en découlent. Ainsi le réseau devient beaucoup plus robuste et si n'importe quelle liaison ou n'importe quel nœud ne devient plus accessible, le réseau restera opérationnel et l'information transitera par d'autres liaisons. Cela a donc pour conséquences directes, l'instauration de protocoles de routage dynamique, capable de prendre des décisions de changement de route de manière quasi-instantanée.

— Décentralisation administrative : Une autre conséquence directe de ce type de topologie et l'absence totale d'entité administrative gérant le réseau et du pouvoir de nuisance potentiel qui lui est associé. N'importe qui a la possibilité de rejoindre le réseau s'il arrive à se connecter à un ou plusieurs nœuds de ce réseau. Il est cependant nécessaire que l'ensemble du réseau se mette d'accord sur les protocoles utilisés (comme IP) et sur l'unicité des ressources dites rares (comme des préfixes d'adresses IP ou des numéros d'AS).

— Une meilleure robustesse face aux congestions : Pour finir, cette topologie possède un grand avantage en matière de gestion des congestions réseau. Pour palier ce problème dans l'approche en étoile, nous devions sans cesse penser à augmenter les capacités du nœud central. Grâce aux protocoles de routages et aux protocoles utilisés, comme les protocoles pair-à-pair, nous pouvons voir une meilleure utilisations des ressources.

Désavantages de la topologie maillée

— Une augmentation exponentielle du nombre de liaisons : Dans une topologie totalement maillée, chaque nouveau nœud doit être connecté directement à tous les autres nœuds du réseau. Par conséquent, pour un réseau à n machines, on se retrouve avec $\frac{1}{2}n^2+\frac{1}{2}n$ liaisons. Cela devient donc très rapidement impossible pratiquement à mettre en œuvre et c'est pourquoi la très grande majorité des réseaux maillés ne le sont pas totalement.

— Un réseau potentiellement incontrôlable : Au delà du fait qu'il n'y ait pas de possibilité

de surveiller tous les échanges du réseau et de surveiller l'agrandissement du réseau, il est essentiel pour le bon fonctionnement du réseau que la totalité des participants soient d'accord sur les protocoles d'interconnexion utilisés et sur la non redondance des ressources rares. Sur l'Internet, la ICANN par le biais de l'IANA fait autorité *de facto* pour l'allocation des ressources. De plus, il devient beaucoup plus facile d'usurper l'identité d'un tiers, en usurpant par exemple des adresses IP ou des numéros d'AS pour TCP/IP.

— Une plus grande complexité : Alors que dans une architecture en étoile, le routage est des plus simples, dans une topologie comme celle-ci le routage pose de réels problèmes d'optimisation. De nombreux travaux de recherche existent sur le routage dynamique.

2.3.3 Topologie du réseau Internet

Ces deux topologies ont une approche radicalement différente et possèdent toutes les deux des limitations lorsque les réseaux grandissent de manière importante. Dans les faits et en particulier sur Internet, les réseaux modernes sont un mélange de ces deux approches où il est bien vu pour la robustesse du réseau de multiplier les connexions directes avec d'autres nœuds sans pour autant pouvoir joindre l'ensemble des nœuds en un saut. Nous allons à présent brosser un peu plus en détails la topologie d'Internet.

Internet est l'interconnexion de plusieurs réseaux d'opérateurs fonctionnant avec la suite de protocole TCP/IP. Pour être opérateur Internet, il est nécessaire de remplir deux conditions :

— S'être vu allouer un numéro de système autonome (*Autonomous System Number* ou *ASN*), numéro unique sur 16 ou plus récemment 32 bits servant à identifier un opérateur.

— S'être vu allouer un ou plusieurs préfixes d'adresses IP (IPv4 ou IPv6). Un opérateur peut également être en charge de préfixes IPv4 de clients qui ne lui appartienent pas et qui lui sont confiés temporairement par un client. On appelle ces préfixes, des préfixes *PI* pour *Provider Independant*.

Ces ressources doivent être uniques sur le réseau et sont déléguées pour Internet par la ICANN/l'IANA à des registres régionaux (RIR) les attribuant eux-mêmes à des registres locaux (LIR). Il existe aujourd'hui plus de 42 000 opérateurs Internet.

Les opérateurs par la suite s'interconnectent entre eux et annoncent à leurs pairs les préfixes IP dont ils ont la charge et éventuellement les préfixes d'autres opérateurs joignables par leur intermédiaire. Le protocole de routage et d'annonces de routes utilisé est le protocole BGP (Rekhter *et al.* (2006)). Plus les interconnexions entre opérateurs sont nombreuses, plus le réseau est robuste. L'objectif pour un opérateur est de pouvoir joindre tous les préfixes de l'Internet et si possible par un maximum de chemins différents. La figure 2.3 est un exemple de différentes interconnexions entre opérateurs.

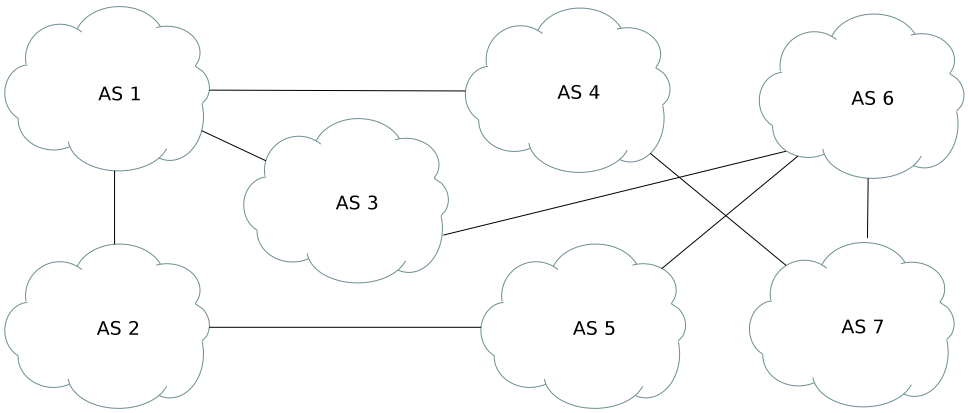

FIGURE 2.3 Exemple de topologie d'interconnexions d'opérateurs Internet.

Chaque opérateur par la suite gère son réseau interne comme il le souhaite. La plupart des grands opérateurs commerciaux fournissant un accès Internet à des particuliers ont un cœur de réseau interconnecté et redondant fonctionnant avec un protocole de routage dynamique, comme OSPF, avec des routeurs de bordures ou de frontières permettant de joindre d'autres AS. Les clients finaux se connectent ensuite par une liaison unique à un routeur connecté au cœur de réseau, appelé BAS ou BRAS (*Broadband* Remote *Access Server*) généralement par le biais d'une connexion DSL sur fils de cuivre ou par le biais du câble pou les particuliers.

2.4 Les modes de communications sur Internet

Après nous être intéressés à la topologie des réseaux, nous allons nous intéresser aux deux grands modes de communication à travers le réseau Internet : l'approche client/serveur et pair-à-pair. Ces modes de communication ne seront pas sans rappeler les topologies que nous avons vu précédemment.

2.4.1 L'environnement client/serveur

Le mode de communication le plus répandu aujourd'hui sur Internet est sans conteste le mode client/serveur. Un ordinateur connecté à Internet possède des ressources qu'il décide de

mettre à disposition d'autres machines que l'on appelle serveur. D'autres machines connectées, désireuses de récupérer le contenu mis à disposition par le serveur, se connectent à ce dernier grâce à son adresse IP et récupèrent la ressource voulue. On appelle ces autres machines des clients. Ce type de fonctionnement, ses avantages et ses défauts, sont très proches de son pendant topologique à savoir la topologie en forme d'étoile. Parmi les protocoles applicatifs les plus populaires d'Internet fonctionnant en client/serveur, on peut citer les protocoles HTTP, SMTP, POP, IMAP, FTP et bien d'autres.

2.4.2 L'environnement pair-à-pair

En parallèle de l'environnement client-serveur, s'est développé l'environnement pair-à-pair proche de la topologie maillée. Dans cet environnement, chaque machine fait à la fois office de client et de serveur, c'est-à-dire qu'elle récupère du contenu tout en le mettant à disposition d'autres machines. Ce type de réseau possède aussi l'avantage de permettre des transmissions de manière directe sans passer par un intermédiaire. Le protocole le plus courant aujourd'hui sont le protocole Bittorent.

2.4.3 Comparaison des deux modèles

Les réseau pair-à-pair possèdent de nombreux avantages comparativement aux réseaux client/serveur parmi lesquels :

— Une meilleure résistance : La charge et le contenu étant distribuées sur tout le réseau, il est beaucoup plus improbable que l'ensemble des machines le constituant devienne indisponible en même temps comparé à un unique serveur ne répondant plus.

— Une meilleure évolutivité : Plus le nombres de machines faisant parti du réseau pair-à-pair est important, plus les ressources mises à disposition sont disponibles puisque chaque nouveau nœud apporte avec lui de la bande passante et un espace de stockage supplémentaire, ce qui permet de répondre à une demande sans cesse de plus en plus croissante. Dans l'approche client/serveur, plus le nombre de clients augmente, plus le serveur devra être capable de supporter la charge associée.

— Une réduction des coûts : Si le réseau pair-à-pair est composé de nœud hébergeant et mettant à disposition gracieusement le contenu comme c'est souvent le cas sur les grands réseaux pair-à-pair, le réseau ne coûtera rien à son concepteur *a contrario* d'un ou plusieurs serveurs en location en salle machine qui peuvent vite devenir coûteux si

les ressources demandées sont populaires.

Il existe cependant des inconvénients à l'approche pair-à-pair dans certaines conditions :

— Une perte de contrôle : Il devient très difficile pour l'émetteur originel de contrôler, d'effacer ou de modifier la ressource mise à disposition une fois cette dernière partagée.

— Des problèmes d'authentification : Sans mécanisme de signature numérique par chiffrement, il est impossible d'attester raisonnablement de l'authenticité du contenu, les nœuds du réseaux ayant la possibilité de l'altérer. Cela est également possible avec une approche client/serveur mais statistiquement moins probable, la quasi-totalité des routeurs des opérateurs ne modifiant pas le contenu des paquets qu'ils acheminent.

— Une perte du contenu : Il n'existe aucune garantie permettant d'affirmer que l'intégralité du contenu partagé à un moment donné est disponible et prêt à être partagé avec un nouveau nœud, en particulier si peu de nœuds composent le réseau.

— Des faibles performances : Si le réseau est composé de peu de nœuds et que ces nœuds ont une bande passante sous-dimensionnée, comme c'est le cas pour des lignes DSL ou câble de particuliers, cela nuira nécessairement au réseau.

2.5 Les réseaux pair-à-pair

Les réseaux pair-à-pair se sont développés en même temps que la démocratisation de l'Internet dans le courant des années 1990. De par leur aspect distribué qui rend leur optimisation complexe, ils ont été sans cesse améliorés au cours des années et se retrouvent dans de nombreuses applications très hétérogènes.

2.5.1 L'évolution des réseaux pair-à-pair

La première génération de protocoles pair-à-pair populaires concernant l'échange de fichiers a démarré avec l'arrivée de Napster (Carlsson et Gustavsson (2001)) en 1999 développé par Shawn Fanning et Sean Parker afin de faciliter en particulier l'échange de fichiers musicaux. Le protocole était basé sur un serveur indexant tous les nœuds du réseau et enregistrant également les fichiers ou morceaux de fichiers mis à disposition par chacun d'entre eux que l'on appelait tracker. Lorsqu'un nœud désirait obtenir une ressource, il faisait une requête à ce serveur qui lui indiquait les nœuds la possédant et étant disponibles pour la lui envoyer.

La vaste majorité des fichiers partagés étant des fichiers sous droit d'auteur échangés sans aucune considération légale, la justice américaine condamna Napster en 2001 suite à une plainte de la RIAA (*Recording Industry Association of America*). Napster obéit à l'injonction de déconnecter son tracker, point central du réseau, ce qui entraîna sa mort immédiate.

C'est suite à cette expérience de déboires judiciaires, et au delà de l'aspect réseautique visant à optimiser les protocoles et l'usage du réseau, que les protocoles suivants ont cherché à éviter au maximum toute centralisation dans leur implémentation en particulier concernant le problème d'indexation. L'idée principale des protocoles de deuxième génération est donc de décentraliser au maximum la base de données d'indexation et de la répartir ou de la dupliquer sur le plus de nœuds possible tout en gardant un réseau efficace et efficient.

C'est ainsi que GNutella a vu le jour et est devenu très populaire. Dans un premier temps, et jusqu'à la version 0.6, le protocole reprend l'idée d'un serveur centralisateur mais redondant. Le nouveau nœud souhaitant se connecter au réseau se connecte à l'un des ces serveurs dont on lui a fourni l'adresse, pour récupérer la liste de tous les autres serveurs. Ces serveurs sont réputés être connectés en permanence. Si un serveur venait à ne plus répondre, les clients effectueraient leur requête à l'un des autres serveurs. Il existait également un mécanisme de synchronisation entre ces différents serveurs pour disposer de la même information. Par la suite, le protocole s'est amélioré en permettant de sélectionner ces serveurs de manière dynamique parmi les nœuds faisant partie du réseau. Ces nœuds sont connus sous le vocable de supernœud et sont choisis en fonction de leur capacité de connexion (débit, gigue, latence) et du la durée de connexion au réseau. Fastrack reprendra ce principe en gardant la hiérarchie des différents pairs et il sera utilisé par des logiciels qui ont été très populaires comme KaZaA entre autre.

Dans la même période, courant 2001, Bram Cohen, ingénieur informatique, crée ce qui est encore aujourd'hui le protocole pair-à-pair le plus populaire : Bittorent. Ce nouveau protocole apporte une avancée majeure comparé à ces prédécesseurs. Le transfert de fichiers, après recherche de nœuds possédant la ressource désirée, ne se fait plus avec un seul pair mais avec plusieurs. Chaque fichier partagé est donc divisé préalablement en plusieurs morceaux appelés *chunk* et pouvant être téléchargés de différentes sources de manière indépendante alors que jusqu'ici cela se faisait en un seul et unique bloc. Il en résulte ainsi, une bien meilleure utilisation de la bande passante disponible du réseau pair-à-pair, puisqu'un pair peut rapatrier dorénavant plusieurs morceaux d'un même fichier provenant de plusieurs pairs en même temps, ce qui est particulièrement efficace sur des connexions Internet asymétriques comme

l'ADSL où la bande passante montante est bien inférieure à la bande passante descendante des différents nœuds. De plus, il est dorénavant possible de partager une partie d'un fichier alors même que la totalité du fichier n'a pas encore été téléchargée. Cela accélère donc de manière drastique les partages de nouveaux fichiers mis à disposition.

2.5.2 Réseaux pair-à-pair structurés et réseaux non-structurés

Les réseaux pair-à-pair sont classés en deux grandes catégories : les réseaux dits non-structurés et les réseaux structurés.

Les premiers réseaux pair-à-pair étaient historiquement non-structurés. Lorsqu'un nœud souhaitait trouver un autre nœud possédant une ressource voulue, celui-ci n'avait aucune idée des nœuds présents sur le réseau possédant potentiellement le contenu. La recherche se faisait principalement au début par inondation pure du réseau. Le nœud demandait ainsi à ses voisins, c'est-à-dire les autres nœuds du réseau dont il connaissait l'existence, s'ils possédaient la ressource désirée. Si la réponse était négative, ces voisins demandaient alors à leurs voisins qui tentaient à leur tour de satisfaire la requête, et ainsi de suite. Potentiellement, si la ressource recherchée n'était pas présente sur le réseau, l'ensemble des pairs aurait reçu une demande la concernant. L'efficience d'algorithmes de ce type, appelés uninformed BFS (Chen *et al.* (2007)), était très mauvaise et d'autres algorithmes meilleurs ont par la suite été proposés, comme le protocole k-Random Walk par exemple. (Lv *et al.* (2002))

Les algorithmes de routage sur les réseaux non-structurés ne sont malheureusement pas très efficaces, puisque les pairs pris individuellement n'ont aucune idée, même approximative, de l'endroit où peuvent se trouver les ressources désirées. *A contrario*, les réseaux structurés permettent de connaître un sous échantillon de pairs d'avantage susceptibles de posséder la ressource. La contrepartie est de devoir imposer des contraintes structurelles plus élevées au réseau.

Ce type de réseau fonctionne la plupart du temps avec un mécanisme de réseau superposé (*overlay network*) où les machines faisant partie du réseau pair-à-pair fonctionnent sur un réseau virtuel logique avec un nouvel adressage qui lui est propre. Ce réseau fonctionne sur le réseau existant, comme Internet par exemple, d'où le nom de "superposé". La plupart des implémentations et des algorithmes utilisés dans les réseaux structurés fonctionnent avec un système de tables de hachage distribuées (*Distributed hash table* ou plus connues sous l'acronyme *DHT*).

2.5.3 Les tables de hachage distribuées

Face au désir de décentraliser la base de données chargée de référencer la localisation et la disponibilité des fichiers sur le réseau, les concepteurs de protocoles pair-à-pair ont proposé un système de répartition de cette base appelée DHT pour tables de hachage distribuées. Chaque fichier ou morceau de fichier se voit associé un hash par le biais d'un fonction de hachage comme md5, sha-1 ou sha-256. En pratique, chaque hash associé est réputé suffisamment unique, c'est à dire que la probabilité d'avoir un hash identique entre deux valeurs différentes est très faible. Les valeurs sont ensuite enregistrées sous la forme d'un couple <clef,valeur> et stockées dans un ordre qui dépend du hash de la clef. La figure 2.4 donne un exemple d'une table de hachage où, à partir d'un nom utilisé comme clef, on réussit à trouver le numéro de téléphone associé grâce au hash calculé à partir de la clef (le nom ici), ces derniers étant classés selon leur hash.

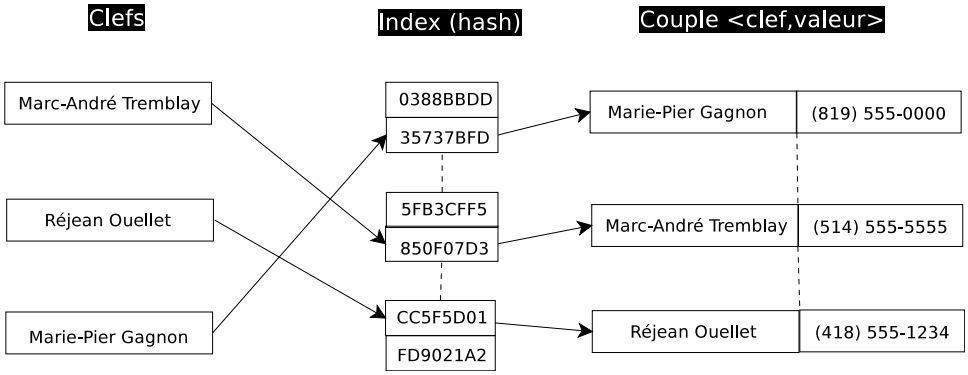

FIGURE 2.4 Exemple de tables de hachage.

Une table DHT reprend donc la notion de table de hachage mais en la distribuant sur plusieurs nœuds. Ainsi, cette dernière se retrouve morcelée sur plusieurs nœuds qui ne possèdent qu'une partie de la table de hachage. Bien évidemment, chaque nœud pouvant se déconnecter à n'importe quel moment, la table de hachage est distribuée de manière redondante. Ce type de réseau pair-à-pair a nécessairement besoin de s'appuyer sur un réseau structuré comme vu précédemment pour joindre facilement et efficacement des nœuds ayant la partie de la table de hachage désirée par le demandeur. De multiples protocoles d'indexation utilisant les DHT ont été proposés et implémentés dans des logiciels pair-à-pair comme CAN, Chord, Pastry ou Tapestry (Stoica *et al.* (2001)). Aujourd'hui, le protocole le plus populaire et que l'on retrouve dans les implémentations récentes notamment de Bittorent se nomme Kademlia.

2.6 Kademlia

Dans le but de créer un réseau dépourvu de toute centralisation, Kademlia propose une sur-couche réseau s'appuyant sur les tables de hachage distribuées vues précédemment. Il a été conçu en 2002 par deux chercheurs Petar Maymounkov et David Mazières (Maymounkov et Mazières (2002)) . Le but de ce protocole est de permettre la localisation d'un nœud ou de ressources sur le réseau grâce à son adresse Kademlia globalement unique. Une métrique de distance est donc instaurée et permet avec l'adresse Kademlia source et l'adresse destination de connaître la distance qui sépare les deux. Cette distance en elle même est totalement arbitraire et n'a aucun rapport avec la distance géographique réelle ou un quelconque nombre de sauts IP. Un noeud peut donc être notre voisin Kademlia alors qu'il est situé à l'autre bout du globe.

Nous allons voir à présent dans cette section comment le protocole Kademlia fonctionne, quelless en sont ses instructions et comment la recherche d'un noeud s'effectue.

2.6.1 L'adressage Kademlia

Chaque noeud, à son arrivée sur le réseau Kademlia se voit attribué une adresse Kademlia. À l'instar des adresses IP qui dépendent du lieu et du fournisseur Internet du nœud, l'adressage Kademlia fonctionne de manière différente. Le nouvel arrivant va générer de lui même et aléatoirement son identifiant sur 160 bits qui sera globalement unique. Il a été montré (Hakiem *et al.* (2012)) que statistiquement, les collisions possibles avec d'autres noeuds s'étant attribués aléatoirement les mêmes adresses sont très rares. Nous pouvons prendre à titre d'exemple la configuration sans état d'IPv6, appelée SLAAC (Thomson *et al.* (2007)), où les adresses IP que s'attribuent les machines fonctionne de la même manière mais sur beaucoup moins de bits (64 bits) et sans encombre. Un mécanisme de vérification permet cependant de s'assurer que l'adresse est belle est bien unique.

2.6.2 La métrique de distance

Cette métrique est le cœur et est la réelle nouveauté du protocole Kademlia comparativement aux protocoles déjà existants. Kademlia se base sur une métrique très simple, connue et qui présente de bonnes propriétés : l'opérateur ou exclusif $\oplus$. L'opérateur $\oplus$ est un opérateur binaire prenant deux opérandes, en l'occurrence les deux adresses Kademlia source et destination dont on souhaite connaître la distance et retourne ladite distance. Les opérations $\oplus$ s'effectuent à un niveau binaire en comparant les bits de chaque adresse un par un.

Cet opérateur à l'avantage de respecter les trois règles fondamentales des distances, c'est-à-dire :

— $d(x, x) = 0$, la distance d'un nœud avec lui-même est nulle.
— $d(x, y) = d(y, x)$, la distance entre deux noeuds est symétrique.
— $d(x, z) \leq d(x, y) + d(y, z)$, correspond à l'inégalité triangulaire, c'est-à-dire que le chemin le plus court entre deux points est la ligne droite. À moins de passer par un autre nœud situé sur cette ligne, la distance totale pour atteindre la destination finale en passant par un autre nœud est plus grande.

  Appliqué aux adresses Kademlia, cela signifie que plus une adresse à ses premiers bits similaires avec l'adresse de destination, plus les deux nœuds sont considérés comme près.

2.6.3 La connaissance du réseau

Chaque nœud a une connaissance limitée du réseau mais peut localiser précisément un autre nœud du réseau à partir du moment où il connaît son adresse Kademlia, le principe de Kademlia étant que chaque noeud possède une meilleure connaissance du réseau dans son entourage. Cela signifie donc que plus la distance entre deux nœuds est faible, plus la probabilité que les deux noeuds se connaissent est grande.

Dès qu'un nœud prend connaissance de l'existence d'un autre nœud du réseau Kademlia, ce dernier va enregistrer dans une base de données son adresse IP et le port sur lequel celui-ci écoute pour pouvoir le joindre plus tard ou pour transmettre ces informations aux autres nœuds qui pourraient les lui demander.

Comme le réseau peut potentiellement être très grand et que chaque nœud a possibilité d'apprendre l'existence d'un nombre très important de voisins, la base de données a un nombre fini d'espaces de stockage prévus à cet effet. Les créateurs de Kademlia ont donc proposé un système sur le principe du *k-bucket*, k étant un nombre entier. Chaque noeud a 160 tables de k entrées de stockage possible. Un nœud a ainsi la possibilité de stocker plus de voisins proches que de voisins éloignés avec la métrique $\oplus$, chaque table permettant de stocker k voisins dont les premiers bits sont similaires à ceux du nœud courant. La première table ne pouvant contenir que $2^0 = 1$ voisin, la deuxième $2^1 = 2$ voisins, la troisième $2^2 = 4$, etc.

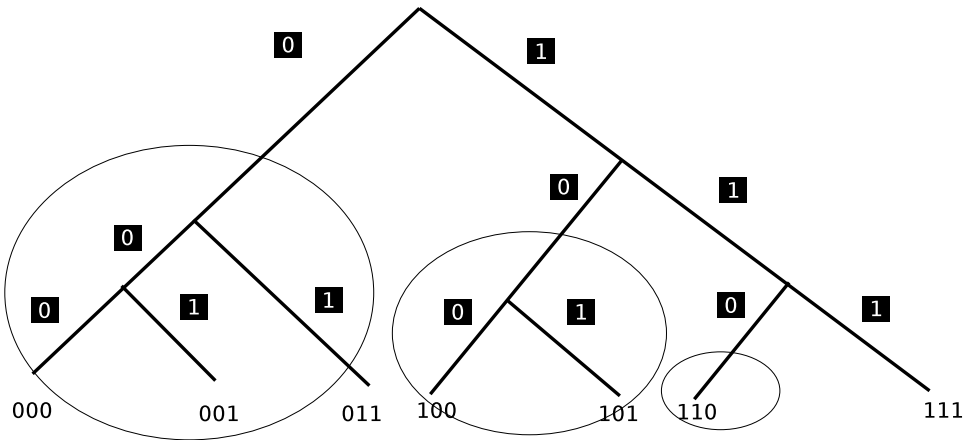

FIGURE 2.5 Exemple de réseau Kademlia et de stockage *k-bucket*.

Prenons un exemple avec la figure 2.5. Dans un but de simplification, nous n'adresserons les nœuds du réseau uniquement avec 3 bits. Dans cet exemple, nous nous mettrons à la place du nœud se trouvant à l'extrême droite de la figure et dont l'adresse est `111`. Le premier bucket que nous allons avoir sera celui dont les deux premiers bits sont identiques à ceux de notre noeud courant, autrement dit tout noeud ayant une adresse correspondant à `11x`. Dans notre exemple, il ne peut nécessairement n'y avoir qu'une seule entrée au maximum puisque la valeur du bit sinon serait égale à celle du noeud courant. Nous stockons donc l'adresse IP et le port d'écoute du nœud ayant l'adresse Kademlia `110` dans ce premier bucket. Il est important de noter cependant qu'il pourrait très bien ne pas avoir de noeud `110` auquel cas ce bucket resterait vide. Nous continuons avec le second bucket qui regroupe cette fois ci tous les noeuds ayant une adresse de la forme `1xy` avec `x` différent de la valeur du deuxième bit de notre nœud courant pour ne pas englober toute la branche, c'est à dire les nœuds ayant une adresse de la forme `10y`. Graphiquement cela correspond aux nœuds compris dans le second cercle. Le raisonnement est le même pour le troisième et dernier cercle comprenant des adresses de la forme `0xy`. Il important de noter que ce n'est pas parce qu'une adresse ne se trouve dans dans le bucket correspondant qu'il n'existe pas de nœud sur le réseau la possédant.

2.6.4 La recherche d'un nœud ou d'une ressource

Comme nous l'avons vu précédemment, un noeud n'a qu'une connaissance partielle du réseau. Il ne connaît, au pire, qu'un seul autre nœud sur le réseau. Par conséquent, quand il va souhaiter joindre un nœud dont il ne connaît *a priori* pas l'adresse, il devra faire des demandes à d'autres nœuds qui seront plus avisés que lui. Si l'on souhaite faire une analogie, on pourrait comparer cela a une cartographie routière. Pour se rendre à Tadoussac depuis Montréal, nous

suivrons dans un premier temps les panneaux Québec, puis la Malbaie pour ensuite suivre Tadoussac. Cela signifie que les nœuds ont une connaissance plus précise de leur voisinage immédiat. Dans Kademlia, nous cherchons des noeuds qui se situent à proximité (toujours avec la métrique ou exclusif) du nœud ou de la ressource recherchée. Nous commençons donc ainsi par demander au nœud que nous connaissons déjà dans notre base de données et qui se situe le plus à proximité du nœud recherché, s'il connaît l'adresse IP et le port associé ou s'il connaît un autre nœud qui est encore plus proche que lui du nœud désiré. Ce type de recherche relève d'une complexité logarithmique en $\theta(\log n)$, nous pouvons donc trouver un nœud rapidement dans le réseau sans que le temps de recherche augmente de manière exponentielle avec la taille du réseau. Si le nœud cherché n'existe pas sur le réseau ou n'existe plus, nous nous retrouverons à un moment de notre recherche avec aucun meilleur voisin pour atteindre le noeud recherché et nous en déduirons que le nœud n'est pas présent. Le fait que les adresses Kademlia soient sur 160 bits, crée *de facto* de grands espaces vides et permet donc d'éviter les attaques de type déni de service, un nœud ne pouvant pas balayer l'espace d'adressage aléatoirement à la recherche de pairs existants.

2.6.5 Les instructions Kademlia

Le protocoles Kademlia possède les commandes suivantes utilisées par les nœuds dans leur recherche :

FIND_NODE(x) Cette commande permet de connaître l'adresse IP et le port d'un nœud dont l'adresse Kademlia `x` est passée en paramètre, si ce nœud existe. La procédure de recherche mise en œuvre est celle décrite dans le paragraphe précèdent.

PING(x) Cette commande permet de savoir si le nœud possédant l'adresse passée en paramètre et joignable. La commande `FIND_NODE(x)` est alors exécutée à moins que l'adresse IP et le port de correspondance soient déjà dans la base de données du nœud. Si cette dernière est concluante, la commande `PING` s'assurera que le nœud demandé est bel et bien présent sur le réseau et ne s'est pas déconnecté entre temps.

STORE(c,v) Le réseau Kademlia étant un réseau fonctionnant à base de DHT sur un réseau structuré comme nous l'avons vu précédemment, il est important que les nœuds partageant un contenu (des *chunks* dans le cas d'un réseau Bittorent) et s'enregistrent auprès des nœuds dont les adresses Kademlia sont les plus proches (au sens de la distance ou exclusif) des hashs des données partagées. Ainsi, ces nœuds responsables d'une partie de la table

d'indexation stockent un couple de données : `c` pour la clef qui correspond au hash de la donnée en question et `v` pour l'adresse et le port du nœud la possédant.

FIND_VALUES(x) Cette commande permet de rechercher un nœud possédant la ressource dont le hash est passé en paramètre. Cette commande va dans un premier temps exécuter la commande `FIND_NODE(x)` avec la même valeur de `x`. Cette commande lui retournera l'adresse IP et le port de la machine la plus proche de cette "adresse". Par la suite la commande `FIND_VALUES(x)` demandera à ce nœud s'il connaît une machine possédant la ressource désirée. Ce nœud retournera les adresses des machines s'étant enregistrées préalablement avec la commande `STORE`.

2.6.6 L'introduction dans le réseau

La phase d'arrivée dans le réseau est importante puisque le noeud doit être capable de joindre les autres noeuds mais il doit également être joignable et donc être connu d'autres nœuds. Comme l'adresse Kademlia d'un nœud est générée aléatoirement par ce dernier, il va par le suite *se placer* sur le réseau, c'est-à-dire qu'il devoir va se faire connaître de ses voisins Kademlia qui ont une adresse proche de la sienne et qui sont déjà connectés. Ainsi ses voisins Kademlia auront connaissance de lui et informeront les pairs souhaitant le joindre et ne connaissant que les voisins du nœud recherché.

Le nouveau nœud doit donc dans un premier temps avoir au moins une adresse d'un autre nœud déjà présent sur le réseau et capable de l'introduire. Ce prérequis est obligatoire pour tous les protocoles pair-à-pair. Pour le protocole Kademlia, le nœud d'introduction n'a pas d'importance à partir du moment que celui-ci connaît aussi déjà un ou plusieurs autres nœuds du réseau.

Après s'être généré soit même son adresse Kademlia de manière aléatoire, le nœud va faire une recherche sur sa propre adresse avec la commande `FIND_NODE(x)` vue précédemment. De cette façon là, il va contacter plusieurs nœuds, dont notamment ses voisins sur la fin de sa recherche, qui vont enregistrer son adresse IP de correspondance et son port d'écoute. Après *cette recherche*, le nœud sera opérationnel sur le réseau et pourra être contacté par d'autres nœuds ne connaissant pas *a priori* directement ce nouvel arrivant.

2.7 Projets similaires

Nous allons a présent nous intéresser à différents projets dans la littérature se rapprochant de ce que nous proposons sans pour autant être identiques et les comparer à ce que nous souhaitons faire.

2.7.1 Projet DIASPORA* et Statusnet

Avec l'avènement des réseaux sociaux, des projets libres ont vu le jour afin d'éviter la centralisation et la concentration des ressources et des données à un même endroit, gérés par un même acteur comme Facebook ou Twitter. Pour palier ce problème, différents projets ont vu le jour. Le pionner dans le domaine se nomme DIASPORA* et se veut être une alternative à Facebook. L'idée principale de DIASPORA* consiste à permettre à n'importe qui d'installer un serveur DIASPORA* nommé *pod* sur son ordinateur ou sur un serveur dédié. L'utilisateur peut par la suite mettre en place son profil sur son *pod* qu'il est le seul à contrôler. Pour pouvoir ajouter des amis ou des contacts, à l'image de Facebook, l'utilisateur entrera le contact avec qui il souhaite se lier sous la forme d'une adresse `utilisateur@poddiaspora`. Tous les utilisateurs ne sont cependant pas obligés d'héberger eux-mêmes leur page, ils peuvent la mettre sur le *pod* d'un ami ou d'une compagnie proposant ce service. Friendica reprendra le même principe que DIASPORA* avec des fonctionnalités supplémentaires et est aujourd'hui une alternative sérieuse à DIASPORA*. Le microblogging est également très populaire et il existe aussi des alternatives décentralisées équivalente à Twitter avec en particulier statusnet basé sur le même concept que DIASPORA*.

Cependant ce type de décentralisation de site web n'est pas une décentralisation générique, elle est appliquée à une application particulière et ne peut donc pas être étendue à n'importe quel site web. L'objectif reste cependant le même pour un domaine particulier et la popularité grandissante de ces réseaux dénote sans conteste un attrait de plus en plus important des internautes pour ce type de protocoles et d'applications.

2.7.2 Projet Bittcoin et NameCoin

Le 24 mai 2009, un développeur répondant sous le pseudonyme de Satoshi Nakamoto diffuse sous la forme d'article de journal un document proposant la création d'une monnaie pair-à-pair totalement décentralisée (Nakamoto (2009)). La monnaie proposée se base sur un système de ressources rares à l'image de l'étalon-or et s'opposant aux monnaies fiduciaires basées sur la confiance. Le modèle proposé eu un certain succès à tel point qu'aujourd'hui le

Bittcoin est présent sur le marché des changes et est très prisé des spéculateurs.

Le protocole proposé permet de disposer une masse monétaire globale fixe, la non reproductibilité d'une unité de valeur et l'échange de Bittcoins sans passer par une quelconque autorité, comme le système bancaire. Dans sa conception, il est très proche d'un étalon or.

Le projet *Dot bit* est un projet basé sur le même système que Bittcoin mais appliqué au système de noms de domaine (DNS) qui dans sa forme actuelle est hiérarchique. *dot bit* propose ainsi un protocole d'annuaire totalement décentralisé en utilisant une extension générique de premier niveau (gTLD) : `.bit`, bien qu'elle n'ait pas été approuvée par la ICANN. À ce jour, ce sont autour de 80 000 domaines complètement décentralisés qui sont en service par ce biais.

Cette idée est très prometteuse et permet de s'affranchir de la ICANN, des registres et des registraires mais n'est pas encore très développée principalement car la branche alternative ne fait pas partie des délégations des serveurs racines ICANN qui sont dans les installations par défaut des résolveurs DNS. Cela prend donc une intervention de l'utilisateur ou de son fournisseur d'accès, ce que nous cherchons à éviter.

2.7.3 Projet P2PWeb

Depuis ces dernières années, la littérature scientifique voit de plus en plus d'articles de projets de décentralisation appliqués au web. Cependant, la quasi-totalité des articles publiés s'intéressent à une décentralisation uniquement dans un contexte de surcharge d'un site web appliqué aux vidéos. Ces travaux sont majoritairement orientés vers la réduction des coûts pour les CDN (*Content delivery network*) comme Akamai et nécessitent l'installation d'un plugin comme firecorall par exemple (Terrace *et al.* (2009)). C'est le cas d'un article publié en 2012 (Ghareed *et al.* (2013)) qui permet aux utilisateurs visitant un site web et disposant d'un plugin associé dans leur navigateur d'accéder en pair-à-pair aux vidéos proposées et compressées en H.264/SVC, le codec H.264/SVC ayant l'avantage de proposer différentes couches de qualité de vidéos et permettant ainsi de regarder des vidéos de manière fluide même avec un débit réduit. Les résultats de ces études mettent en avant des résultats intéressants sur l'efficience de tels protocoles en comparant le nombre de pairs sur le réseau (typiquement des nœuds ayant une faible bande passante montante) et les métriques classiques de qualité de service. Ces résultats nous montre qu'à partir d'un nombre relativement faible de pairs, le réseau permet des débits montants tout à fait corrects pour une utilisation classique : 25 clients recevant du contenu à 300 ko/sec pour 48 pairs partageurs. Le fait d'installer un logiciel

côté client allant contre nos contraintes que nous nous sommes fixées, la façon de procéder est différente mais l'objectif reste le même et par conséquent ces résultats ne peuvent que nous encourager à continuer dans cette voie.

2.7.4 Projet WebRTC

En mai 2011, Google annonce son projet de développer une API (*Application programming interface*) permettant d'étendre les fonctionnalités des fureteurs web actuels en permettant de nouvelles possibilités de communications par la voix et la vidéo ainsi que l'échange de fichiers en pair-à-pair et sans l'aide d'un plugin (Group (2012)). Le projet a été nommé WebRTC (Web Real-time Communication) et a pour but de répondre aux évolutions d'usage des utilisateurs. Par la suite, le W3C, autorité en charge de normaliser les langages du web, a décidé de se l'approprier pour en faire en standard. Un *draft* a commencé à être rédigé en septembre 2012 et est toujours à ce stade en juillet 2013. Pour l'instant le projet est expérimental et n'est supporté que par quelques navigateurs. Les premiers résultats semblent cependant très prometteurs et les possibilités de développement très vastes. À noter par exemple, des projets comme Peer-CDN (Wu *et al.* (2008)) qui prévoient d'alléger la charge des serveurs en faisant du partage vidéo pair-à-pair avec les fureteurs compatibles.

Notre projet se rapproche de ces idées. Le projet n'étant qu'à l'étape de brouillon et peu compatible avec l'ensemble des fureteurs, il n'est pas encore opérationnel. De plus, le projet se concentre d'avantage sur l'aspect multimédia et n'envisage à aucun moment une décentralisation complète de site web.

CHAPITRE 3

STRATÉGIE DE DÉCENTRALISATION PROPOSÉE

Dans ce chapitre, nous proposerons donc une solution au problème de distribution de sites web en pair-à-pair en détaillant le fonctionnement de notre modèle.

3.1 Définitions

Avant de détailler notre solution, nous définissions les termes que nous utiliserons par la suite :

— client : Ce terme est à comprendre dans le sens des architectures client/serveur. Un client se connecte à un serveur afin d'y récupérer une ressource, un serveur web dans notre cas.

— partageur : Un partageur est une machine connectée à Internet faisant à la fois office de serveur web HTTP et de serveur DNS pour les clients en plus d'être un nœud actif du réseau pair-à-pair pour l'échange des ressources à partager avec les autres partageurs.

— partageur de référence : Le partageur de référence est le partageur qui a émis originellement le contenu à partager.

— site web partagé : site web dont les serveurs sont des partageurs.

— ressource : fichier hébergé par les partageurs et potentiellement désiré par les clients. Cela peut par exemple être des pages HTML, des images ou bien des feuilles de style. L'ensemble des ressources forment un site web.

— servent : machine opérant à la fois comme serveur et comme client. Les nœuds des réseaux pair-à-pair sont des servents.

3.2 Hypothèses

Dans cette section, nous énonçons différentes hypothèses sur lesquelles nous nous appuierons pour la suite de notre étude. Ces hypothèses se veulent proche de la topologie d'Internet et de l'usage qui en est fait aujourd'hui ou dans les années à venir :

— On suppose que les clients disposent d'un fureteur web et d'un résolveur DNS ou sont configurés pour en contacter un.

— On partira du principe que les clients auront le même comportement qu'avec n'importe quel autre site web, c'est-à-dire qu'ils ne doivent pas installer de nouveaux navigateurs, de plugins ou d'add-ons pour communiquer avec des sites web partagés.

— Le site web partagé possède un nom de domaine pleinement qualifié et n'est pas joignable uniquement par son adresse IP.

— La très grande majorité des partageurs seront réputés n'avoir qu'un faible débit montant (*upload*) comparativement au débit descendant (*download*) dans les proportions que l'on trouve typiquement sur les connexions ADSL ou câble.

— Les partageurs seront réputés avoir une connexion pleine et entière à Internet, c'est-à-dire qu'ils seront réputés être joignables. L'adressage IPv4 arrivant à saturation, la totalité des opérateurs Internet migrent en IPv6 et dans les années à venir de plus en plus d'appareils seront connectés avec le nouveau protocole, abolissant ainsi les systèmes de type NAT Masquerade bridant la connectivité.

— Les partageurs proposent gracieusement leur ressource en les mettant à disposition. Par conséquent, il est important de pouvoir prendre en considération leurs préférences quant à leur degré de partage.

— Afin de faciliter notre approche, nous nous placerons dans le contexte de sites non dynamiques. Nous excluons donc les forums ou des blogues avec des sections commentaires.

— Nous ne considérerons pas non plus les questions de sécurité pour les mêmes raisons.

3.3 Description sommaire de la solution

Afin de répondre à notre sujet de recherche, nous allons proposer que les partageurs possèdent plusieurs serveurs applicatifs interagissant entre eux de façon à répondre de manière transparente aux demandes des clients. Ces serveurs seront : un serveur web, un serveur DNS faisant autorité, un servent pair-à-pair et une base de données de type SQL. Ces serveurs tout en respectant les RFC spécifiant les échanges entre les clients et les serveurs, agiront cependant différemment de la manière conventionnelle quant à la façon de les traiter.

La figure 3.1 est un résumé de notre approche. Sur la partie gauche de la figure se trouve les partageurs et le partageur de référence. Ils forment à eux, un réseau pair-à-pair permettant l'échange des différentes ressources du site web partagé. Sur la partie droite, on notera deux clients web cherchant à accéder à la ressource et se connectant à l'un des nœuds du réseau pair-à-pair. Il est très important de comprendre qu'à n'importe quel moment un client, de la partie droite du schéma peut devenir un partageur et passer à gauche juste en installant un logiciel respectant les spécifications que nous allons développer.

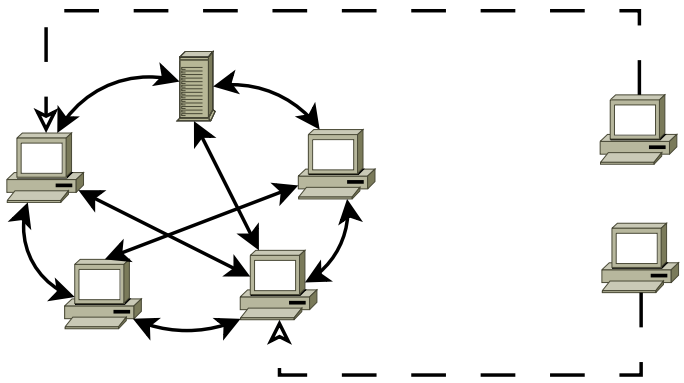

FIGURE 3.1 Approche sommaire de la solution

Pour respecter nos hypothèses nous utiliserons principalement le protocole DNS pour rechercher des partageurs possédant les ressources désirées et nous essayerons de répartir au maximum la charge en distribuant les téléchargements par des modifications au niveau du serveur web. Nous proposerons également des optimisations dans la gestion des échanges entre partageurs. Pour finir nous décrirons un processus d'élection pour incorporer notre système dans le système de noms de domaine ICANN.

3.4 Différents acteurs du réseau

3.4.1 Client

Le client, lors de la demande d'une page web comme `www.example.com` va dans un premier temps interroger son résolveur DNS pour connaître l'adresse IP du domaine recherché. Ce dernier fera une résolution comme vue précédemment dans notre revue de littérature et retournera au client l'adresse IP en question. Par la suite, le navigateur du client va se connecter à l'adresse IP récupérée pour demander la page désirée. Si la page désirée contient des ressources externes, ce qui est pratiquement toujours le cas aujourd'hui, il va les récupérer en faisant éventuellement de nouveau une résolution de nom de domaine si ces ressources se trouvent sur un autre domaine. Les types de ressources classiques sont généralement des images, des fichiers de formatage de style (CSS) ou bien des script Javascript.

Sans modification du côté du client, ce qui contreviendrait à nos hypothèses citées précédemment, nous ne pouvons pas espérer et supposer d'avantage du comportement du client.

3.4.2 Partageur

Les partageurs devront donc prendre en considération ces contraintes. En ce qui les concernent, puisqu'ils acceptent sciemment de partager le site web et ont fait le choix d'installer un ensemble de logiciels, nous ne nous focaliserons uniquement sur ces derniers où nous nous disposons d'une marche de manœuvre plus importante.

Ainsi, chaque partageur disposera des fonctionnalités suivantes :

— Un serveur web : ce serveur écoutera sur le port 80 et sera chargé de répondre aux requêtes des clients se connectant à sa machine pour récupérer des ressources. Selon le type de ressource demandée, il les fournira au client s'il les possède et qu'il a suffisamment de débit disponible restant ou bien il redirigera le client vers un autre partageur. S'il s'avère qu'il retourne effectivement la ressource demandée, il pourra éventuellement la modifier dans un objectif de répartition de charge.

— Un serveur DNS : ce serveur est un serveur faisant autorité sur une zone donnée. Il sera chargé de retourner les adresses IP des partageurs mettant à disposition les ressources demandées par un client de la façon la plus optimale pour le réseau de partageurs. Il s'agit de la pièce angulaire de notre système.

— Un servent pair-à-pair d'échange de fichiers : ce servent a pour objectif de permettre l'échange des ressources formant le site web entre les différents partageurs. Les utilisateurs partageurs étant réputés partager leurs ressources sur une base volontaire, il est important que le servent prenne en compte les contraintes spécifiques que l'utilisateur aura définie. Dans un but de décentralisation complète, le protocole d'indexation et de recherche qui sera utilisé sera le protocole Kademlia que nous avons eu l'occasion de voir précédemment dans la revue de littérature.

— Une base de données SQL : cette base de données sera chargée de stocker l'ensemble des informations dont le partageur a besoin. Elle servira également au partage des ressources entre les différents serveurs. Cela pourrait être le cas par exemple pour le serveur web et le servent pair-à-pair sur les ressources réellement enregistrées et disponibles sur la machine du partageur ou bien entre le serveur DNS et Kademlia afin de connaître l'adresse IP d'un autre partageur présent sur le réseau.

3.5 Préférences du partageur

Le partageur mettant à disposition une partie de ses ressources pour le bien de la communauté, il est important qu'il puisse régler soit même le degré de partage qu'il souhaite concéder. En plus d'entraîner des frustrations et une mauvaise réputation, des partageurs n'ayant plus suffisamment de ressources pour leurs usages, cela serait également néfaste pour le réseau puisque cela entraînerait un nombre important de départs et donc affaiblirait le réseau du site web partagé. Par ressources, nous entendons donc deux ressources principales :

— L'espace disque alloué par le partageur pour le site et plus généralement pour le logiciel. Il est cependant primordial d'avoir des partageurs disposant de suffisamment de place disponible pour accueillir les plus gros fichiers qui ne pourront pas être partagés dans les réponses aux requêtes HTTP des clients.

— Le débit maximal alloué par l'utilisateur autant bien débit montant que descendant. Idéalement, ce paramètre devrait correspondre aux capacités réelles de la connexion Internet du partageur. Cela n'est en réalité que peu pénalisant pour l'utilisateur puisque, parallèlement à cela nous définirons des priorités de traitement des flux réseaux.

3.6 Stratégie de répartition de la bande passante

Dans nos hypothèses de départ, nous supposons que la quasi-totalité des partageurs sont des ordinateurs individuels derrière une connexion Internet dite "classique", c'est-à-dire une connexion Internet DSL ou câblée. Malheureusement la plupart de ces connexions sont des connexions asymétriques et le débit maximum descendant est bien inférieur au débit maximum montant. Cela est particulièrement problématique pour un serveur qui utilise bien plus son débit montant que descendant et c'est ce qu'utiliserons nos partageurs en grande partie lors des réponses aux différents clients. Le seul moment où les partageurs seront amenés à utiliser leurs bande passante descendante étant lors de la récupération des différentes ressources par le biais du réseau pair-à-pair.

Il est donc primordial, dans un soucis d'efficience pour répondre dans un délai raisonnable aux clients, de distribuer au maximum le téléchargement des ressources auprès des différents partageurs. Comme nous nous sommes fixés comme contrainte pour les clients de ne disposer d'aucun logiciel supplémentaire implémentant des comportements spécifiques de leur côté, nous n'avons pas la possibilité technique pour un client de télécharger une ressource donnée sur plusieurs partageurs en même temps, les clients s'attendant à récupérer l'entièreté de chaque ressource qu'ils demandent. Cependant, comme la quasi-totalité du contenu web aujourd'hui présent sur Internet possède des liens externes vers d'autres types de fichiers, comme des images, des fichiers CSS externes, des fichiers Javascript ou bien des pages incluses dans d'autres, nous allons en profiter pour répartir la charge au maximum en délégant le téléchargement par les clients de ces fichiers sur d'autres partageurs, ci ce dernier est surchargé. Cela a donc pour conséquence que toutes les demandes de page HTML seront analysées et modifiées à la volée selon les paramètres du réseau de partageurs et que nous détaillerons par la suite.

3.7 Le serveur DNS autorité

Chaque partageur a un serveur DNS autorité répondant aux requêtes sur le port 53 avec les protocoles TCP et UDP. Ce serveur fait autorité sur le domaine du site partagé et permet de distribuer la charge sur l'ensemble du réseau. Prenons l'exemple d'un site web disponible à l'adresse `www.example.com` et d'un partageur possédant l'adresse IP `2001:db8::1`. Notre serveur fera donc autorité sur le domaine `example.com` et répondra à toutes les demandes de résolution de noms terminant par `example.com`.

Avant de détailler le traitement de chaque type de requête, nous allons définir une fonction à double sens permettant de convertir une adresse IP en une suite ASCII et inversement. Cette fonction est en réalité une fonction de conversion entre des nombres en base 16 (nous utiliserons des adresses IPv6) et des nombres en base 42 utilisant la partie des caractères ASCII utilisable dans des adresses FQDN. Chaque adresse IPv6 est composée de 32 chiffres hexadécimaux. Le nombre de chiffres dont nous aurons besoin en ASCII sera tel que :

$$42^n > 16^{32} \Leftrightarrow n > \frac{\log_2(16^{32})}{\log_2(42)} \Leftrightarrow n > \frac{4^{32}}{\log_2(42)} \Leftrightarrow n > 23,737 \tag{3.1}$$

Par conséquent, nous aurons besoin de 24 chiffres en base 42 pour transcrire ces adresses IP. Nous avons fait le choix des chiffres suivants rangés dans l'ordre croissant :

$$0; 1; \cdots 8; 9; a; b; \cdots; y; z; A; B; \cdots; Y; Z \tag{3.2}$$

Nous définissons ainsi deux fonctions IP2ASCII et ASCII2IP transformant respectivement une adresse IP donnée en paramètrem en une série de caractères ASCII et une séries de caractères ASCII en une adresse IP.

Ainsi, dans notre exemple, après avoir calculé `IP2ASCII(2001:db8::1)`, nous obtenons l'adresse ASCII suivante : `1ErCxE9llrzr1usbBCBarq0x`.

Le serveur DNS est notre outil permettant de déléguer de manière transparente des requêtes à d'autres partageurs que nous allons détailler ici. Dans un premier temps, nous verrons comment techniquement nous permettons une délégation d'une requête à un autre partageur alors que dans un second temps, nous verrons à quel moment intervient la décision au niveau DNS de déléguer la requête.

3.7.1 Délégation d'une requête à un autre partageur

Si le serveur DNS reçoit des requêtes de résolution de noms de domaine sous la forme `www.ascii.example.com` avec `ascii` une adresse IPv6 convertie en adresse ASCII, cela signifie que le serveur doit déléguer la réponse à à la machine disposant de l'adresse ASCII passée en paramètre de la requête. Même si l'adresse ASCII/IP ne correspond à aucune machine connectée ou à un partageur du site, le serveur autorité fera une délégation tant que le format de l'adresse ASCII est valide, par soucis d'efficacité. Le serveur répondra par une réponse qui variera selon l'adresse ASCII du nom de domaine demandé :

— S'il ne s'agit pas de l'adresse ASCII correspondant à l'adresse IP du partageur interrogé pour la résolution, la réponse sera une délégation au partageur à qui appartient l'adresse ASCII de la zone `ascii.example.com`. La réponse sera donc de la forme :

```
1ErCxE9llrzr1usbBCBarq0y IN NS ns.1ErCxE9llrzr1usbBCBarq0y
```

le résolveur cherchera ensuite à résoudre le nom de domaine à qui l'on vient de déléguer la zone, c'est-à-dire `ns.1ErCxE9llrzr1usbBCBarq0y.example.com`. La réponse sera celle d'un enregistrement de type glue (*glue record*) retournant l'adresse IP correspondant au domaine demandée :

```
ns.1ErCxE9llrzr1usbBCBarq0y IN AAAA 2001:db8::2
```

À partir de ce moment là, la requête a été complètement déléguée à `2001:db8::2` qui la traitera selon sa propre décision.

— S'il s'agit de l'adresse ASCII du partageur interrogé, la requête lui est donc adressée et il lui est demandé d'y répondre dans la mesure de ses capacités. À ce stade, il n'a pas possibilité de savoir s'il dispose de la ressource demandée puisque le client ne le précisera qu'une fois la connexion HTTP établie, il ne peut donc que prendre une décision sur son débit restant disponible pour satisfaire de nouvelles demandes HTTP. Nous verrons par la suite à partir de quand un partageur se considère surchargé de requêtes. Selon la réponse à cette question, le serveur DNS du partageur répondra de l'une de ces deux façons :

  — Il estime qu'il est en mesure de satisfaire de nouvelles requêtes, il répond donc par une réponse de type AAAA en spécifiant son adresse. Dans notre exemple, la réponse serait :

  ```
  www.1ErCxE9llrzr1usbBCBarq0y IN AAAA 2001:db8::2
  ```

  Les requêtes DNS s'arrêtent donc là et le client se connectera à `2001:db8::2` pour demander la ressource qu'il désire.

  — Ou bien, il estime ne pas être en mesure de pouvoir répondre et recherche dans

sa base de données des partageurs successibles de pouvoir répondre à sa place. Idéalement, il aura eu l'occasion d'obtenir ces informations là précédemment comme nous le verrons. Sinon, il devra utiliser le protocole Kademlia pour trouver un partageur potentiellement disponible pour satisfaire la requête. Une fois le nouveau partageur trouvé, on lui déléguera la requête par le biais d'un CNAME :

```
www.1ErCxE9llrzr1usbBCBarq0y IN CNAME www.ascii2.example.com.
```

3.7.2 Traitement d'une résolution de type www.example.com

L'adresse du site web en question est et reste `www.example.com`. Le client demandera donc à résoudre cette adresse pour accéder au site. Il ne s'attend pas non plus à voir sa barre d'adresse changer en `www.1ErCxE9llrzr1usbBCBarq0z.example.com` à chaque fois qu'il passe d'une page à une autre du site. En plus de ne pas être très esthétique, cela serait néfaste pour le réseau si l'utilisateur venait à partager cette adresse puisque la seule machine capable de répondre à des requêtes sur ce domaine est `2001:db8::3` et n'est pas supposée être joignable en permanence.

Toute requête afin de résoudre `www.example.com` sera considérée par le partageur chargée d'y répondre comme une requête lui étant adressée et il s'interrogera de la même façon que vue précédemment pour savoir s'il est en mesure de répondre favorablement ou non à la demande :

— S'il prend la décision de répondre à la demande, il répondra :

```
www IN AAAA 2001:db8::3
```

— Sinon, après avoir trouvé l'adresse d'un autre partageur (`ascii` ici) pouvant le faire, il répondra :

```
www IN www.ascii.example.com.
```

3.7.3 Gestion des TTL

La gestion des champs TTL (*Time to live*) expliquée lors de la revue de littérature est cruciale pour notre modèle. En effet, les partageurs ont un comportement imprédictible et peuvent quitter le réseau à tout moment, par conséquent, si une résolution DNS effectuée

précédemment est encore valide et que l'adresse du partageur en mémoire est un partageur qui n'est plus connecté, cela entraînera une indisponibilité du service à moins de vider complètement le cache du résolveur DNS. Nous devons donc dans certain cas fixer le TTL a la valeur la plus faible possible. La norme (Mockapetris (1987)) ne fixe aucun minimum, nous considérerons donc un TTL à 0 seconde.

3.7.4 Résumé de la gestion DNS

Nous pouvons résumer le comportement de notre serveur DNS dans les tableaux suivants. Il est important de noter que tous les partageurs ont le même comportement. Dans la suite, `ascii` est considéré comme l'adresse ASCII d'un partageur quelconque. `2001:db8::1` et `ascii2` sont respectivement les adresses IPv6 et ASCII du partageur interrogé. Le partageur répondra différemment selon le scénario :

— Dans le premier scénario, présenté par le tableau 3.1, `ascii` $\neq$ `ascii2` et le partageur peut répondre à de nouvelles demandes HTTP.

TABLEAU 3.1 Comportement du DNS d'un partageur non surchargé (cas 1)

Requête	Type	Réponse	TTL
`www.example.com`	`AAAA`	`2001:db8::1`	0
`ascii.example.com`	`NS`	`ns.ascii.example.com`	∞
`ns.ascii.example.com`	`AAAA`	adresse IPv6 de `ascii`	∞
`www.ascii2.example.com`	`AAAA`	`2001:db8::1`	0

— Voyons maintenant le cas, présenté dans le tableau 3.2, où `ascii` $\neq$ `ascii2` mais que le partageur ne peut plus répondre à de nouvelles demandes HTTP. `ascii3` est l'adresse d'un autre partageur capable de potentiellement y répondre à sa place.

TABLEAU 3.2 Comportement du DNS d'un partageur surchargé (cas 1)

Requête	Type	Réponse	TTL
`www.example.com`	`CNAME`	`www.ascii3.example.com`	0
`ascii.example.com`	`NS`	`ns.ascii.example.com`	∞
`ns.ascii.example.com`	`AAAA`	adresse IPv6 de `ascii`	∞
`www.ascii2.example.com`	`CNAME`	`www.ascii3.example.com`	0

— Voyons à présent le cas, présenté dans le tableau 3.3 où `ascii` = `ascii2`, autrement dit que le partageur interrogé est celui a qui le client veut demander la ressource. Plaçons nous dans le cas où nous pouvons y répondre.

TABLEAU 3.3 Comportement du DNS d'un partageur non surchargé (cas 2)

Requête	Type	Réponse	TTL
`www.example.com`	`AAAA`	`2001:db8::1`	0
`ascii.example.com`	`NS`	`ns.ascii.example.com`	∞
`ns.ascii.example.com`	`AAAA`	`2001:db8::1`	∞
`www.ascii.example.com`	`AAAA`	`2001:db8::1`	0

— Enfin, dans le tableau 3.4, toujours en gardant `ascii` = `ascii2`, mettons nous dans le cas où le partageur ne peut plus répondre à de nouvelles demandes HTTP. `ascii3` est l'adresse d'un autre partageur capable de potentiellement y répondre à sa place.

TABLEAU 3.4 Comportement du DNS d'un partageur surchargé (cas 2)

Requête	Type	Réponse	TTL
`www.example.com`	`CNAME`	`www.ascii3.example.com`	0
`ascii3.example.com`	`NS`	`ns.ascii3.example.com`	∞
`ns.ascii3.example.com`	`AAAA`	adresse IPv6 de `ascii3`	∞
`www.ascii.example.com`	`CNAME`	`www.ascii3.example.com`	0

Nous venons ainsi de voir la gestion des requêtes DNS par les différents partageurs et le mécanisme mis en place afin de décentraliser en pair-à-pair notre site web sans autorité centrale tout en permettant de garder le processus transparent pour les clients.

3.8 Le serveur web HTTP

En plus d'avoir un serveur DNS, tous les partageurs disposent d'un serveur web HTTP écoutant sur le port 80 en TCP et permettant de satisfaire les requêtes des clients. Tout comme le DNS, le comportement de notre serveur HTTP sera différent d'un serveur HTTP standard mais respectera les standards HTTP cités dans la RFC (Fielding *et al.* (1999)) notamment ceux détaillés dans la revue de littérature.

3.8.1 Enregistrement de la topologie du site

Lors du processus de récupération des différentes ressources à partagées et récupérées par le réseau pair-à-pair comme nous le verrons plus tard plus en détail, le serveur web se doit de procéder à un enregistrement de la topologie partielle du site qu'il dispose afin de répondre de manière efficace aux requêtes des clients une fois le moment venu. La topologie que nous utiliserons est une topologie sous forme de graphes et où chacun des sommets représente

une ressource. Pour chaque sommet nous enregistrons plusieurs informations listées dans le tableau 3.5 et dont l'on trouve deux exemples à droite.

TABLEAU 3.5 Caractéristiques des sommets du graphe d'un site partagé

Champs	Champs	Champs
identifiant	42	48
type MIME	text/html	image/png
localisation	/example.html	/images/image.png
taille	30 ko	67 ko
priorité	12	5
liens externe[]	[12, 48, 57]	—

Voyons plus en détail la signification de chacun des champs du talbeau 3.5 :

— *identifiant* : Chaque ressource/sommet dispose d'un identifiant unique qui permet aux autres sommets de l'identifier.

— *type MIME* : renseigne le type MIME de la ressource enregistrée. Selon le type MIME, le comportement suite à la demande d'une ressource sera différent.

— *localisation* : correspond à la localisation sur le serveur par rapport à la racine. La ressource peut ensuite être récupérée en la téléchargeant avec l'adresse web : `http://www.example.com/{localisation}`.

— *taille* : la taille en ko de la ressource. Cette information nous est importante pour optimiser la répartition du téléchargement par les clients que nous verrons par la suite.

— *priorité* : correspond à un nombre entier. À chaque fois qu'un client demande la ressource, ce compteur est incrémenté.

— *liens externes* : Ce dernier paramètre est un paramètre contenant des *identifiants* définis précédemment. Tous les *identifiants* listés sont des ressources à lequel la ressource courante fait référence et qui devront être téléchargées aussi par le client dans la foulée. Ce champ ne peut être renseigné uniquement si le type MIME de la ressource est de type `text/html` ou `text/css`, les autres types (images et Javascript) ne pouvant faire de références à d'autres.

L'ensemble de la topologie est stockée dans une base de données SQL de notre serveur de base de données.

3.8.2 Gestion des requêtes des clients

Le serveur web écoute les connexions entrantes et attend que des clients viennent lui demander des ressources constituant le site web qu'il partage. On supposera donc que si les clients se connectent au partageur, c'est que précédemment le serveur DNS autorité de ce même partageur n'a pas délégué à un autre partageur le traitement du client et que par conséquent il est en mesure de répondre à cette demande.

Deux cas de figures se présentent lors d'une demande :

— Si le partageur n'a pas la ressource désirée, il va en déléguer le traitement à un autre partageur la possédant. En effet, alors que nous ne pouvons *a priori* pas savoir quelle ressource sera demandée par le partageur au moment de la résolution DNS, cela n'est pas le cas au moment de la requête HTTP puisque le client demande explicitement la ressource qu'il souhaite. Le serveur après vérification, interroge sa base de données contenant la liste de différents partageurs possédant la ressource ou à défaut interroge le serveur pair-à-pair par le biais du protocole Kademlia. Une fois qu'il a récupéré l'adresse d'un partageur possédant la ressource en question, il va répondre au client non pas par un code `HTTP 200 OK` mais par un code `HTTP 302 FOUND` dont la réponse tient en deux lignes. Par exemple, pour la ressource `42`, le serveur répondrait ainsi, s'il souhaite déléguer le traitement de cette requête au partageur ayant l'adresse `2001:db8::1`.

```
HTTP/1.1 302
Location: http://www.1ErCxE9llrzr1usbBCBarq0x.example.com/example.html
```

Ceci aurait pour effet immédiat pour le fureteur du client de contacter `2001:db8::1` après avoir fait la résolution DNS pour lui demander la page qu'il possède et ainsi satisfaire la requête du client. Le partageur ayant délégué la requête prendra conscience que cette ressource est une ressource demandée par les clients et par conséquent il en informe le serveur pair-à-pair qui en augmentera la priorité de récupération comme nous verrons un peu plus tard.

— Si le partageur est malgré tout surchargé il agira de la même façon que décrit précédemment

à la différence que s'il ne connaît pour le moment aucun partageur possédant la ressource demandée, il ne va pas faire de recherche mais déléguer le traitement de la recherche a un autre partageur de sa liste afin de ne pas l'encombrer encore d'avantage. Le client interrogera donc un autre partageur qui n'a probablement pas la ressource désirée et qui fera lui la recherche d'un partageur qui la possède.

— Si le partageur n'est pas surchargé et dispose effectivement de cette ressource, il va la retourner au client en opérant éventuellement à des modifications du contenu comme nous allons le voir.

3.8.3 Envoi des ressources aux clients

Dans le cas où le serveur HTTP a pris la décision de répondre à la demande, il va interagir différemment selon le type de ressource demandé :

— S'il s'agit de fichiers qui structurellement ne font pas référence vers d'autres fichiers, alors le serveur web les envoie directement sans les modifier comme un serveur HTTP classique le ferait. C'est le cas de tous les formats d'images (`png`, `jpg`, `gif`, `svg`...) mais aussi des fichiers Javascript (`js`) qui ne peuvent pas faire référence à d'autres fichiers non plus.

— Dans le cas d'un fichier faisant référence à des ressources externes comme les pages HTML (`html`, `htm`) ou des feuilles de style externes (`css`), nous effectuerons un pré-traitement avant de renvoyer la ressource. Ce pré-traitement consiste à procéder à une décentralisation autant que possible de la récupération du contenu par le client en modifiant à la volée les adresses de références vers d'autres ressources que le client aura besoin de récupérer pour afficher l'entièreté de la page et qu'il téléchargera donc par la suite. Ces adresses se retrouvent pour le langage (X)HTML au niveau des balises suivantes :

  — `<img src="..." />` spécifiant l'affichage d'une image donnée en paramètre.
  — `<link rel="stylesheet" href="..." />` précisant l'adresse d'une page de style externe.
  — `<iframe src="..." />` ou bien `<frame src="..." />` entraînant l'affichage d'une page dans une autre.
  — `<script src="..." />` pour la localisation du fichier contenant des scripts Javascript.

Pour les fichiers CSS :

— `@import ...;` faisant référence à un autre fichier externe à inclure dans le fichier externe.

— `url(...);` faisant référence à une ressource à télécharger, comme par exemple une image de fond.

Ces paramètres seront donc analysés grâce à un analyseur lexical au moment de la récupération de ces ressources par pair-à-pair. Avant d'effictivement enregistrer ces zones comme des zones où nous pourrons modifier les adresses à la volée et que nous appellerons par la suite "zones injectables", nous devons nous assurer que les références sont bien des références faites vers des ressources du site que nous souhaitons partager et non pas vers un site tiers qui assumera la charge de son côté, le cas échéant. Cela se fait de manière très simple en vérifiant que l'adresse de cette ressource telle que spécifiée, est un chemin relatif et non pas absolu. Si l'adresse de la ressource commence par `http://` , nous ne considérons pas cette zone comme injectable.

Les zones injectables sont stockées dans notre base de données afin d'être consultées à chaque requête de client sur ces ressources.

TABLEAU 3.6 Caractéristiques des zones injectables

Champs
Identifiant
Ressource
RessourceExterne
offset

Voyons la signification de chacun de ces champs :

— *Identifiant* : identifiant unique pour chaque ressource analysée.

— *Ressource* : ressource correspond au champ défini précédemment dans notre indexation du site web sous forme de graphe. Ce numéro doit donc correspondre à une ressource de type `text/html` ou `text/css`. Le couple *identifiant*/*ressource* est unique.

— *RessourceExterne* : ce champ correspond à la ressource externe qui sera demandée

dans la zone injectable. Elle correspond également à l'identifiant utilisé dans notre processus d'indexation du site web mais est étendu à toutes les ressources.

— *offset* : correspond à l'emplacement où se trouve la zone injectable.

Lors de la demande d'une ressource possédant des zones injectables, le serveur s'interrogera sur la meilleure répartition possible et que nous détaillerons par la suite. Il a ainsi plusieurs possibilités qui s'offrent à lui. Prenons l'exemple de la ressource `image.png` auquel la page `index.html` fait référence par une balise de type `<img/>` :

— Il peut décider de prendre à sa charge le téléchargement du fichier `image.png` et retourne ainsi au client la balise `<img src="image.png" />`. Même si l'enregistrement DNS qui a servi pour atteindre le partageur est déjà expiré (nous avons fixé le TTL à 0), le client va quand même récupérer ladite ressource auprès du partageur qui lui retourne la page grâce aux connexions de type `Keep-Alive`. Le client ne fermera pas la connexion TCP courante après la récupération de `index.html` et demandera directement `image.png` dans la foulée.

— Il peut décider de prendre la décision de ne pas retourner lui même au client la ressource en question et va donc déléguer cette tâche à un autre partageur (`2001:db8::2` par exemple ici). Il écrira ainsi la balise de la manière suivante : `<img src="http://www.1ErCxE9llrzr1usbBCBarq0y.example.com/image.png" />`. Le client une fois la page `index.html` récupérée interrogera `2001:db8::2` pour récupérer l'image. Il est intéressant de noter que la récupération peut se faire en cascade.

3.9 Servent pair-à-pair

En plus des serveurs DNS et HTTP, nous utiliserons un servent (à la fois un client et un serveur) pair-à-pair afin de permettre la communication et l'échange de fichiers entre les différents partageurs. Le servent permettra aux nouveaux partageurs de télécharger les ressources qu'ils mettront à disposition des clients et de les partager par la suite aux autres partageurs. Au fur et à mesure des connexions des clients, les partageurs seront aussi amenés à télécharger les ressources que les clients ont demandées mais n'avaient pas au moment de la demande (et qui ont donc envoyé un `HTTP 302 FOUND`). Le servent pair-à-pair utilisera le protocole de recherche et d'indexation Kademlia que nous avons vu tout à l'heure. Contrai-

rement aux autres serveurs, le servent pair-à-pair aura presque le même comportement qu'un servent pair-à-pair classique dans la récupération des ressources à quelques exceptions qui sont listées ci après :

— Ajouter un champ d'un entier à tous les partageurs dans leurs caractéristiques afin de leur permettre de renseigner leurs homologues s'ils sont capables d'accepter de nouvelles connexions (s'ils ne sont pas surchargés) et à quel débit (en kbps) le transfert se réalisera. Pour indiquer qu'il n'est pas en mesure d'accepter de nouvelles connexions, le partageur mettra ce champ à `0`, sinon il indiquera le débit maximal avec lequel il répondra aux requêtes.

— Permettre la consultation par les serveurs HTTP et DNS des partageurs ayant les ressources demandées, leur disponibilité et le débit offert le cas échéant.

— Permettre de gérer quelles ressources récupérer et quelle en devrait être la priorité de récupération par consultation de la base de données.

Pour se connecter au réseau pair-à-pair, le nouveau partageur fera une requête, tel un client classique, vers la ressource `http://www.example.com/tracker.torrent`. Le nouveau partageur contactera donc un autre partageur pour récupérer un fichier d'introduction au réseau pair-à-pair dont le nœud d'introduction au réseau est le même partageur qui lui a envoyé le fichier.

3.10 Gestion de la bande passante

Nous l'avons vu précédemment à de multiples reprises, les différents serveurs applicatifs sont amenés à savoir si le partageur est en capacité de pouvoir répondre à de nouvelles demandes provenant de clients et s'ils sont surchargés, de déléguer cette charge à un autre partageur. Dans cette section nous allons voir comment le partageur répond à cette question et de manière générale, comment la bande passante du partageur est gérée.

3.10.1 Répartition et réservation de débit

Tous les partageurs sont réputés connaître leur capacité maximale approximative d'envoi que nous nommerons x par la suite. Nous partons également du principe que cette valeur n'est pas trop changeante et que la connexion est relativement stable. Pour récupérer la valeur de x, nous utiliserons un serveur distant comme par exemple notre partageur de référence, réputé

être hébergé en salle serveur avec une importante connectivité. Nous mettrons également x à jour régulièrement par observation du nombre de paquets TCP perdus lors des différentes demandes de clients.

Il est important de se rappeler que le partageur est réputé être un ordinateur individuel que l'utilisateur utilise pour ses besoins personnels et qu'il ne fait que consentir à mettre son ordinateur à disposition pour utiliser la bande passante qu'il n'utilise pas. Par conséquent, il est très important que notre ensemble de serveurs ne s'accapare pas le débit disponible et donc que l'utilisateur ne subisse pas de lenteur de connexion due à notre programme, ce qui entraînerait probablement une suppression du programme et une fragilisation du site web partagé.Le partageur ne fera qu'une estimation du débit auquel il s'attifera la demande, estimation qui sera d'avantage vue à la baisse. Pour cela, nous définissons plusieurs paramètres :

— *pourcentUtilisation* : Pourcentage maximal d'utilisation réputée de la capacité maximale x.

— *DebitMinimal* : Débit minimal pour qu'un partageur puisse répondre à des demandes. En bas de cette valeur, le partageur sera considéré comme surchargé.

— nombreRequetesEnCours : nombre de requêtes HTTP en cours de satisfaction au moment de la demande.

Le partageur partira donc du principe qu'il dispose à tout moment de $x*pourcentUtilisation$ kbps pour répondre aux demandes de clients aussi bien DNS que HTTP ou d'échange de ressources entre partageurs. Cela est peut être en réalité supérieur ou inférieur à cette valeur, si x n'est pas suffisamment à jour.

Lors de la demande de satisfaction d'une requêtes HTTP, le partageur va procéder à un simple calcul pour savoir s'il est en mesure ou non d'y répondre favorablement :

$$\frac{x * pourcentUtilisation}{nombreRequetesEnCours + 1} > DebitMinimal \tag{3.3}$$

Si cette inégalité est vérifiée, alors le partageur répondra positivement à la demande en

informant le partageur demandeur qu'il enverra la ressource à un débit de :

$$\frac{x * pourcentUtilisation}{nombreRequetesEnCours + 1} \tag{3.4}$$

Aucun client n'est prioritaire par rapport a un autre et si de nouveaux clients viennent récupérer une ressource auprès du partageur auquel l'un d'eux était déjà connecté, cela se fera à son détriment.

3.10.2 Priorisation

Il est intéressant de noter que nous n'avons pas pris en compte le débit montant du partageur sur le réseau pair-à-pair dans le même temps. Cela est en réalité dû au mécanisme que nous proposons d'utiliser, de différencier des classes de service par le biais de la Qualité de Service (QoS). Nous allons donc distinguer différents flux et les prioriser en fonction de leur contenu. La liste ci-dessous classe les différents flux que nous gérerons par ordre du plus prioritaire au moins prioritaire :

— Les applications tierces. Elles sont réputées être lancées par l'utilisateur et donc sont les plus prioritaires.

— Les réponses aux requêtes DNS des résolveurs des clients. Notre système se basant principalement sur ce mécanisme, il est important qu'il puisse fonctionner à sa capacité maximale en permanence d'autant qu'elles ne génèrent que très peu de volume.

— Les requêtes Kademlia faites par le servent pair-à-pair pour la recherche des ressources sur le réseau de partageurs.

— Les réponses aux requêtes HTTP des clients par les partageurs. Ce sont les requêtes les plus consommatrices comme nous l'avons vu.

— L'envoi des ressources sur le réseau pair-à-pair aux autres partageurs ne les ayant pas.

Une application dite prioritaire peut ainsi, si elle demande un débit supérieur à celui non utilisé par les autres applications, s'accaparer le débit utilisé par les autres applications moins prioritaires qu'elle. De même, son débit se voit réduit si une application ayant une priorité plus importante à la sienne en fait la demande.

3.10.3 Répartition de récupération des ressources

Nous l'avons vu, le serveur HTTP est amené à prendre des décisions sur l'injections d'URL lorsque les ressources injectables (`text/html` et `text/css`) lui sont demandées. La décision est prise de façon à répartir la charge sur plusieurs partageurs en même temps afin de permettre un rendu plus rapide pour les clients.

Lorsqu'une ressource injectable lui est demandée, le partageur recherche dans un premier temps dans sa base de données s'il connaît des partageurs possédant des ressources liées à la ressource demandée et, si tel est le cas, il contacte ces différents partageurs pour connaître le débit auquel ils pourraient satisfaire la demande et qui est calculé comme nous l'avons vu précédemment. Si ce n'est pas le cas, il utilise le protocole Kademlia pour faire une demande auprès du réseau pair-à-pair de partageurs pour connaître des partageurs susceptibles de posséder la ressource.

Une fois la liste des partageurs récupérée, auquel le partageur demandeur s'ajoute s'il dispose également de la ressource en question, le partageur va faire en sorte de minimiser le temps de téléchargement total, tout en minimisant le nombre de partageurs satisfaisant les demandes du client.

3.10.4 Indexation préventive

L'interrogation des partageurs par le protocole Kademlia est une procédure souffrant d'un temps de latence très important (Kumar *et al.* (2005)), par conséquent il est nécessaire d'éviter à tout pris de faire ces recherches au moment où un client fait une demande. Chaque partageur va donc chercher à indexer les autres partageurs susceptibles de posséder les ressources qu'il serait amené à distribuer pour une meilleure efficacité.

Ainsi, dès le démarrage du partageur, ce dernier va rechercher les autres partageurs possédant les ressources qu'il serait susceptible d'injecter dans les ressources injectables dont il dispose pour les stocker en mémoire dans sa base de données. Ces adresses devront être gardées à jour le plus possible, mais il se peut qu'un partageur quitte le réseau pair-à-pair sans prévenir et que le partageur demandeur se retrouve à devoir faire ses demandes auprès des partageurs restants.

Lorsque le serveur HTTP souhaite consulter la liste des partageurs ayant une ressource

souhaitée, il va ainsi récupérer la liste en mémoire et contacter chacun de ces partageurs directement afin de leur demander le débit qu'ils sont prêt à allouer. Le partageur demandeur en profitera pour enlever de sa base toutes les adresses de partageurs ne répondant plus et il lancera également une recherche Kademlia, s'il se retrouve avec peu de partageurs restant, et dont les résultats serviront à remplir la base de données pour les prochaines requêtes de clients.

3.11 Gestion de l'espace disque

Nous l'avons vu précédemment, les partageurs donnent la possibilité à leur propriétaire de régler le volume d'espace disque qu'ils sont prêts à concéder à l'application. Dans la suite, nous nommerons ce volume par la variable y. Chaque partageur, lors de la première installation ne possède aucune ressource et va télécharger au fur et à mesure des demandes des clients, les ressources voulues sur le réseau pair-à-pair. Ainsi les premières demandes de clients auprès d'un nouveau partageur entraînerons nécessairement des réponses de redirection `HTTP 302 FOUND` et auront pour effet d'incrémenter le compteur *priorité* de la ressource demandée dans notre graphe. Les servents des partageurs téléchargent toutes les ressources dont le champ *priorité* est non-nul et avec la priorité correspondant au champ. Ainsi uniquement les ressources demandées sont téléchargées et parmi elles, les plus populaires sont téléchargées plus rapidement. Nous pouvons cependant être confronté à un problème d'espace :

— Dans le cas où la taille totale du site web est inférieure à y, nous n'avons pas à choisir entre les différentes ressources puisque toutes peuvent être stockées sur les disques du partageur.

— Si la taille totale du site est supérieure à y, nous serons amenés à faire un choix lorsque le partageur n'aura plus de place disponible pour stocker de nouvelles ressources. Si tel est le cas, nous étendrons le principe de popularité aux ressources conservées. Seules les ressources ayant un compteur *priorité* important seront conservées. S'il n'y a plus de place disponible sur le partageur et qu'une ressource est plus populaire que la ressource la moins populaire stockée, mais n'est pas présente sur le disque, alors le partageur supprimera la ou les ressources les moins populaires pour avoir suffisamment d'espace disque disponible sur le disque pour l'accueillir.

3.12 Gestion des DNS dans Internet

Chaque partageur est susceptible de recevoir des requêtes DNS de résolution de nom de domaine. Un problème se pose cependant : comment un client fait-il pour contacter un premier partageur pour résoudre le domaine du site ? Dans la hiérarchie DNS telle que nous l'avons vue dans la partie II, les résolutions se font en partant d'un serveur racine qui délègue à d'autres serveurs des sous-domaines qui peuvent à leur tour déléguer des sous-sous-domaine.

Les registres responsables des TLD (ccTLD, gTLD) proposent pour leur quasi-totalité, la possibilité de déléguer des sous-domaines (comme `example.com`) à plus de dix serveurs DNS autorité différents. Un résolveur DNS pourra ainsi contacter l'un de ces dix serveurs pour résoudre ses requêtes sur le domaine voulu. Par un mécanisme de *round robin*, chacun de ces serveurs autorité sont contactés à tour de rôle afin d'équilibrer les charges. De plus, les registres ne permettent pas une mise-à-jour trop fréquente de cette liste en plus d'ajouter une valeur de TTL élevée. Les spécifications du protocole DNS obligent un résolveur DNS à contacter un autre serveur DNS autorité si celui contacté ne répond pas. La résolution du nom se fera quand même mais introduira un délai supplémentaire qui est à éviter.

Nous allons donc devoir introduire une discrimination entre nos différents partageurs pour en choisir dix parmi eux qui seront promus et ajoutés à cette liste. La décision sera prise par le partageur de référence à l'origine du partage du site et propriétaire du domaine. Une fois sa décision prise, il informera de lui même le registre de sa sélection ou de sa mise-à-jour, par le biais de son registraire.

3.12.1 Processus d'élection

Il est donc primordial, pour l'intégrité du réseau de choisir les partageurs ayant la probabilité la plus faible de quitter le réseau et ceux qui ont un important débit montant. Des machines en data-centre seraient l'idéal et pourraient exclusivement être dédiées à cette tâche pour une multitude de sites partagés. Nous allons définir les constantes suivantes fixées par le partageur de référence et qui seront utilisées pour la prise de décision :

— *dureeMinimale* : durée minimale en minute de temps de connexion pour être considéré comme éligible.

— *debitMimum* : débit minimum en Ko par seconde pour être considéré comme éligible.

— *debitMaximum* : débit maximum en Ko par seconde considéré lors de la prise de décision.

— *ponderation* : pondération entre la durée de connexion et le débit.

Nous définissions aussi les variables suivantes :

— *debit* : débit (Ko/s) du partageur considéré par notre calcul.

— *duree* : durée (minutes) du partageur considérée par notre calcul.

Nous allons utiliser un postulat statistique (Kumar *et al.* (2005)) qui a montré de manière empirique que plus une personne est connectée depuis longtemps, plus la probabilité qu'elle soit encore connectée dans une heure est importante. Nous pondérerons cependant ce postulat par le débit offert par ces partageurs. Pour être un bon élu, nos partageurs doivent être capables de répondre à une charge plus importante que les autres partageurs. Le partageur de référence calcule ainsi tous les scores des partageurs et sélectionne les dix meilleurs. S'il souhaite lui même être parmi eux, il se sélectionne également sans élection et ne choisira que neuf autres partageurs. Le score d'un partageur est défini comme tel :

— si $debit < debitMimum$ ou que $duree < dureeMinimale$, le partageur n'aura aucun score et ne sera pas considéré comme éligible.

— sinon nous calculerons le score suivant après avoir éventuellement fixé *debit* à la valeur de *debitMaximum*, si celui si est supérieur :

$$duree * ponderation * debit \tag{3.5}$$

Une fois les meilleurs sélectionnés, il envoie au registre leurs adresses pour mise-à-jour. Les adresses envoyées sont de la forme `ns.1ErCxE9llrzr1usbBCBarq0x.example.com` et devront également être couplées à un enregistrement de type glue sur le domaine en question. Ce processus d'élection devrait avoir lieu le plus régulièrement possible tout en évitant de se faire bannir par le registre pour cause d'inondation (quelques fois par jour au maximum).

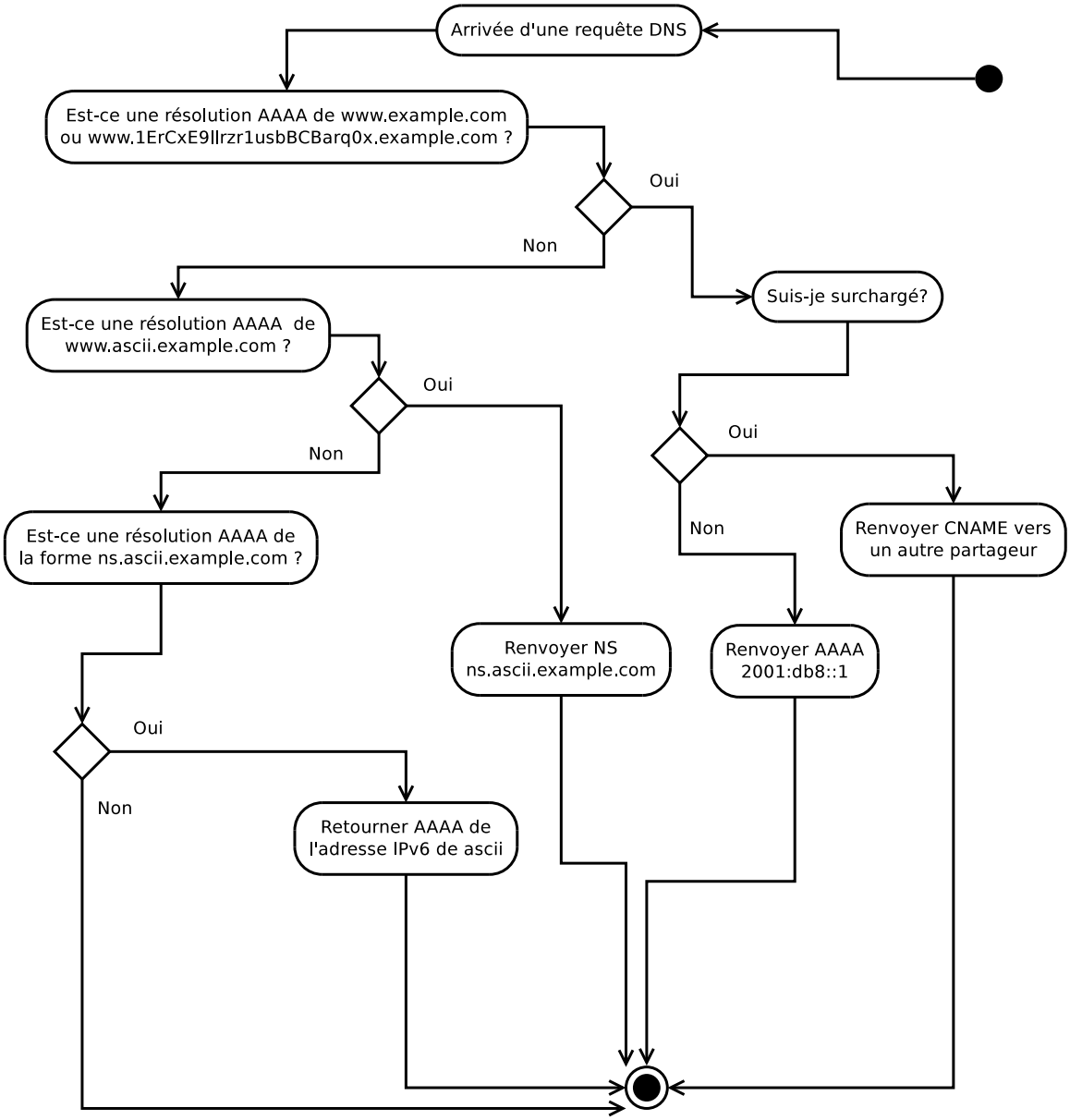

FIGURE 3.2 Diagramme d'activité du comportement DNS de notre modèle avec un partageur ayant l'adresse 2001 :db8 : :1

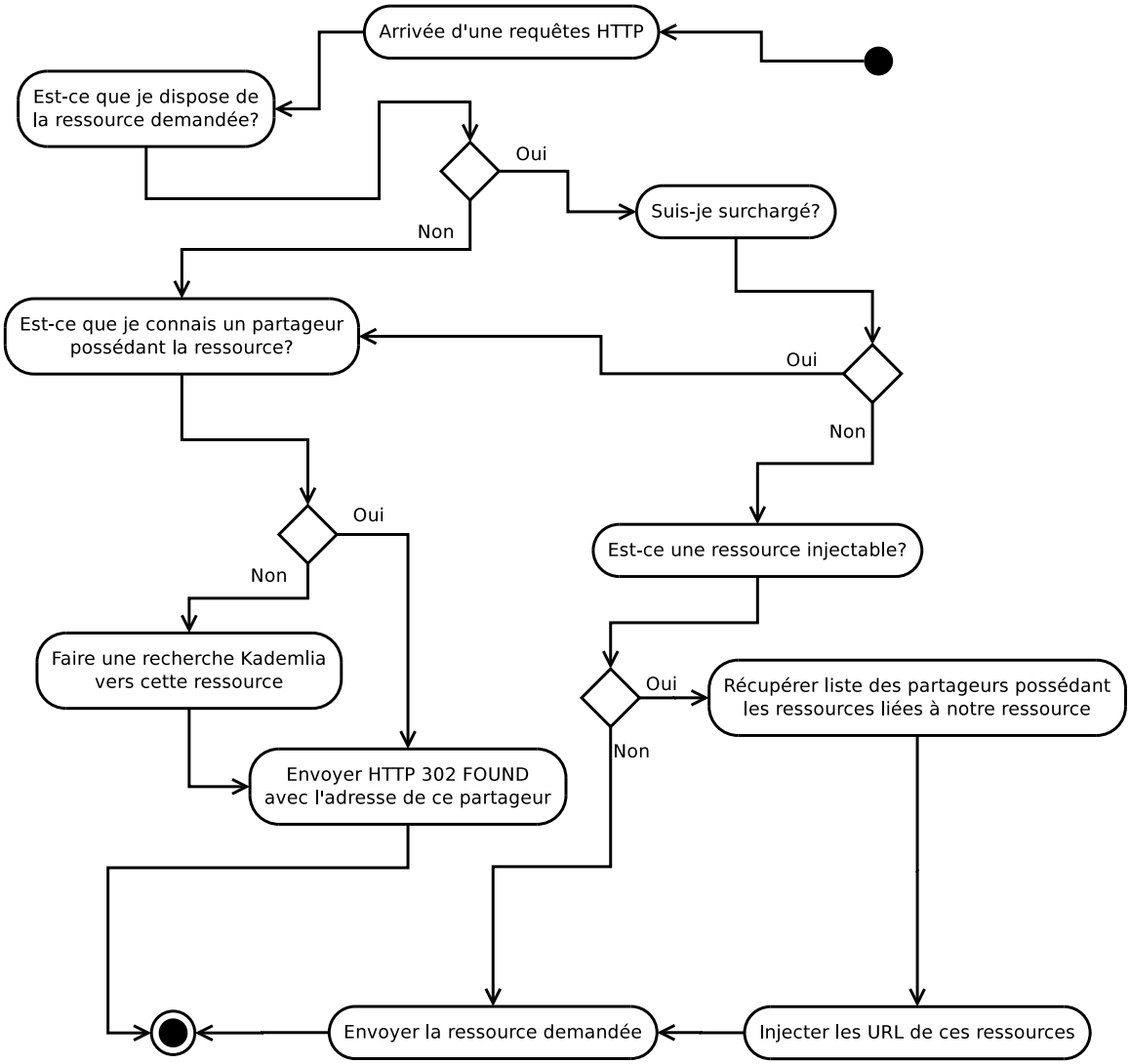

FIGURE 3.3 Diagramme d'activité du comportement HTTP de notre modèle

CHAPITRE 4

ANALYSE DE PERFORMANCE

Après avoir passé en revue les différentes technologies et projets sur lesquelles nous sommes appuyés, nous avons présenté un prototype d'un ensemble de serveurs permettant de réaliser une décentralisation transparente de sites web. Dans cette dernière partie, nous allons en mesurer la performance sur plusieurs critères et la comparer avec le modèle client/serveur classique. Dans un premier temps nous procéderons à une comparaison sur les débits offerts entre le modèle client/serveur et notre modèle pour, par la suite, nous pencher sur la disponibilité offerte pour le service.

4.1 Définition des critères de performance

Avant de procéder à une comparaison des différents modèles, nous allons dans un premier temps définir les critères sur lesquels nous nous appuierons par la suite. Les métriques que nous chercherons à comparer entre les deux modèles seront les suivantes :

— $(g : n, c \rightarrow D)$ Débit moyen observé (D) pour satisfaire tous les clients (c) en fonction du nombre de partageurs (n). Cette fonction permet de comparer les cas de congestion de sites web dans les différentes approches et voir si l'une des solutions est plus robuste que l'autre. Nous comparerons ces résultats avec une une charge équivalente sur un serveur en salle machine.

— $(h : n \rightarrow P)$ Probabilité (P) d'un site web de ne plus être disponible en fonction du nombre de partageurs (n). Nous comparerons les résultats aux SLA (*Service Level Agreement*) des locateurs de serveurs en *data-centre*.

4.2 Valeurs de nos modèles

Afin de mesurer ces différents critères, nous allons devoir fixer certains paramètres sur lesquels nous nous appuierons par la suite et que nous utiliserons de manière identique dans les deux modèles (site web partagé et approche client/serveur). Ces valeurs se veulent le plus proche possible de ce que l'on trouve effectivement sur Internet :

— Le serveur en salle machine aura un débit symétrique de 100 Mbps.

— Les partageurs disposeront en moyenne de 5,16 Mbps de débit montant et de 18,38 Mbps de débit descendant en utilisant les statistiques de milliers de mesures parmi les connexions des différents pays de l'OCDE provenant du rapport final pour l'année 2012 du *berkman center for internet and society at Harvard University* agrégeant entre autre les données du site Internet Speedtest.net ayant des statistiques sur des millions de connexions ou Akamai, le plus gros CDN (*Content Delivery Network*) au monde [1].

— Les partageurs seront connectés principalement en journée et en particulier en début de matinée et fin de journée.

Nous allons dans un premier temps comparer notre modèle avec l'approche client / serveur d'un point de vue analytique. Afin de simplifier notre modèle, nous avons émis quelques hypothèses :

— Le partageur de référence dispose d'une connexion symétrique en 100 Mbps.

— Les partageurs disposent des ressources demandées par les clients ce qui devrait souvent être le cas après un certain temps d'activité du réseau de partageurs.

— Les partageurs auront des débits montants et descendants suivant une loi normale centrée réduite. Les valeurs utilisées pour ces courbes seront :

TABLEAU 4.1 Débits maximums

	Débit montant	Débit descendant
Espérance (μ)	5,16	18,38

4.3 Débit constaté

Voyons tout d'abord, avec une approche analytique, les débits que nous sommes susceptibles de constater pour nos deux modèles. Pour faire cette comparaison, nous définissons tout d'abord la taille d'une ressource possédant des ressources externes, typiquement une page HTML avec des images. Par la suite, nous comparerons le débit théorique pour un

1. http ://cyber.law.harvard.edu/pubrelease/broadband/data.html

nombre donné de clients en fonction du nombre de partageurs.

Nous utiliserons pour cette simulation, les valeurs suivantes, correspondant aux valeurs moyennes enregistrées sur un échantillons représentatifs aux États-Unis et présentées avec le tableau 4.2.

TABLEAU 4.2 Débits maximums

Type de machine	Débit montant (kbps)	Débit descendant (kbps)
Serveur	100 000	100 000
Partageur	1 526	6 637

De même nous définirons deux pages HTML avec plusieurs dépendances que nous listerons avec le tableau 4.3 et correspondant à la page d'accueil du site de l'IETF (`www.ietf.org`).

TABLEAU 4.3 Page de l'IETF et ses dépendances

Type de ressource	Taille (ko)
Page HTML	24
Image #1	13
Image #2	18
Image #3	7
Image #4	6
Page CSS	8

Nous supposerons que les clients du site web ou du site web partagé ont des débits descendants équivalent aux débits descendants des partageurs puisque ce sont *a priori* le même type de machine.

Scénario client/serveur

Dans ce scénario, nous utiliserons la formule suivante pour calculer le débit moyen dans le scénario du serveur :

$$debit = \min(ddc, dms/x) \tag{4.1}$$

Avec :

— ddc : le débit descendant des clients, correspondant à la valeur définie par le tableau 4.2.

— dms : le débit montant du serveur défini par le tableau 4.2.

— x : le nombre de clients simultanés à un moment donné.

Scénario d'un site web partagé

Pour ce qui est de notre modèle, nous ne pouvons malheureusement pas modéliser de la même manière le débit associé. Il s'agit en réalité d'un problème d'optimisation combinatoire relevant d'une complexité NP-difficile que l'on pourrait ramener à un problème de sacs à dos multiples. Pour prendre cette décision, nous aurons besoin des paramètres suivants :

— dmp, le débit montant des partageurs.

— n, le nombre de partageurs possédant les ressources voulues.

— x, le nombre de clients demandant la page et ses dépendances.

— Les différents fichiers et leur taille en ko.

Comme tout problème de cette complexité, nous devrions utiliser des heuristiques si l'espace de recherche se révèlve trop important. Dans notre exemple, cependant, nous nous contenterons d'explorer l'espace de recherche entier afin de trouver la meilleure solution par un algorithme de type procédure par séparation et évaluation (*branch and bound*). Nous utiliserons de faibles nombres de clients et partageurs pour cela. Dans un second temps, nous utiliserons une heuristique afin d'explorer des espaces de recherche plus grand.

Résultats

Nous avons donc décidé de faire un calcul intermédiaire afin de trouver le nombre de kilooctets maximal que nous pourrions paralléliser. Une fois ce nombre obtenu, nous calculerons par équivalence le débit moyen de téléchargement total pour les clients par proportionnalité.

Pour ce faire, nous exécutons notre programme en faisant varier le nombre de partageurs et le nombre de clients simultanés dont les résultats se trouvent dans le tableau 4.4 et le diagramme 4.1. Nous décidons ensuite dans le tableau 4.5 et le diagramme 4.2 de calculer le débit total offert par notre réseau de partageurs selon leur nombre et le nombre de clients. Nous utilisons pour cela les valeurs de partageurs définies préalablement. Au delà de cinq partageurs et trois clients, nous observons déjà une explosion combinatoire et nous n'avons obtenu les autres valeurs que par déduction pour les scénarios à 5 partageurs et 4 et 5 clients.

TABLEAU 4.4 Volume de données parallélisables pour la page d'accueil de l'IETF (en ko)

Clients / Partageurs	1	2	3	4	5
1	76	152	228	304	380
2	38	76	114	152	190
3	26	51	76	102	133
4	24	38	57	76	95
5	24	31	...	62	76

TABLEAU 4.5 Débits observés sur le réseau en fonction du degré de parallélisation pour la page d'accueil de l'IETF (en kbps)

Clients / Partageurs	1	2	3	4	5
1	1526	1526	1526	1526	1526
2	3052	3052	3052	3052	3052
3	4460	4548	4578	4548	4360
4	4832	6104	6104	6104	6104
5	4832	7482	...	7482	7630

Comparons ces résultats aux résultats du tableau 4.7 correspondant à la somme des différents débits maximaux des partageurs. Nous pouvons le constater et c'est tout à fait normal que cette valeur est toujours atteinte lorsque le nombre de partageurs est le même que le nombre de clients puisque chaque partageur ne servira qu'une fois chaque ressource différente permettant à tous les partageurs de terminer le transfert en même temps. Cela n'est par contre pas le cas pour tous les transferts où certains partageurs sollicités termineront leurs transferts plus tôt que d'autres, bien que nous évitons autant que possible que cela arrive. Le diagramme 4.3 illustre le taux d'utilisation de la bande passante disponible des différents partageurs de notre réseau en fonction des clients et du nombre de partageurs. Comme nous pouvons le constater le taux d'utilisation pour cette page est bon et s'approche de 100 %. La seule exception notable est lorsque nous sommes en présence d'un seul client et un nombre important de partageurs. Cela est dû à un seuil que nous atteignons lors de la phase de parallélisation où nous nous pouvons pas aller plus loin puisque nous serons toujours limité par la récupération du plus gros fichier étant donné que nous ne pouvons pas segmenter les différents fichiers.

Pour finir, intéressons nous au débit de récupération pour les clients pris individuellement, ce que nous cherchons à maximiser pour offrir une meilleure qualité d'expérience à

TABLEAU 4.6 Débit observé pour un client donné en fonction du degré de parallélisation pour la page d'accueil de l'IETF (en kbps)

Clients / Partageurs	1	2	3	4	5
1	1526	763	508	382	305
2	3052	1526	1017	763	611
3	4460	2274	1526	1137	872
4	4832	3052	2034	1526	1221
5	4832	3741	...	1871	1526

l'utilisateur final qu'une simple récupération de tous les fichiers auprès d'un seul partageur. Le tableau 4.6 et le diagramme 4.4 présentent ces résultats en kilo-bits par seconde. Nous n'avons ici pas considéré une quelconque limitation du côté de la bande passante descendante du client. Bien évidemment plus le nombre de clients augmente, plus le débit servi par client diminue mais pas de manière proportionnelle puisque nous distribuons au maximum la charge. Les débits servis peuvent être estimés comme raisonnable pour une utilisation par un internaute bien que cette perception dépende de chaque individu.

TABLEAU 4.7 Capacité maximale théorique du réseau de partageurs (kbps)

Partageurs	
1	1526
2	3052
3	4578
4	6104
5	7630

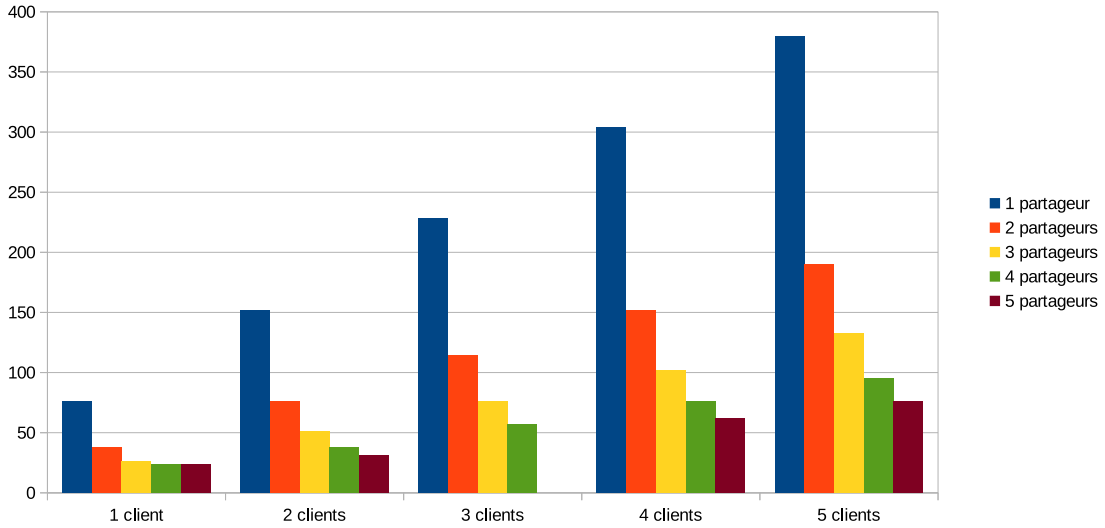

FIGURE 4.1 Volume de données parallélisables (en ko)

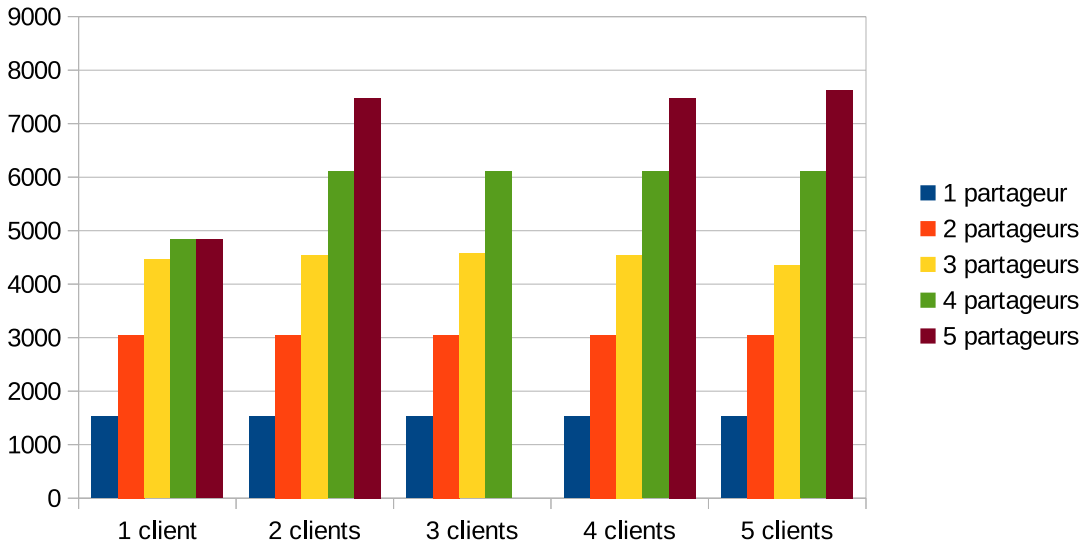

FIGURE 4.2 Débit total utilisable par le réseau de partageurs (en kbps)

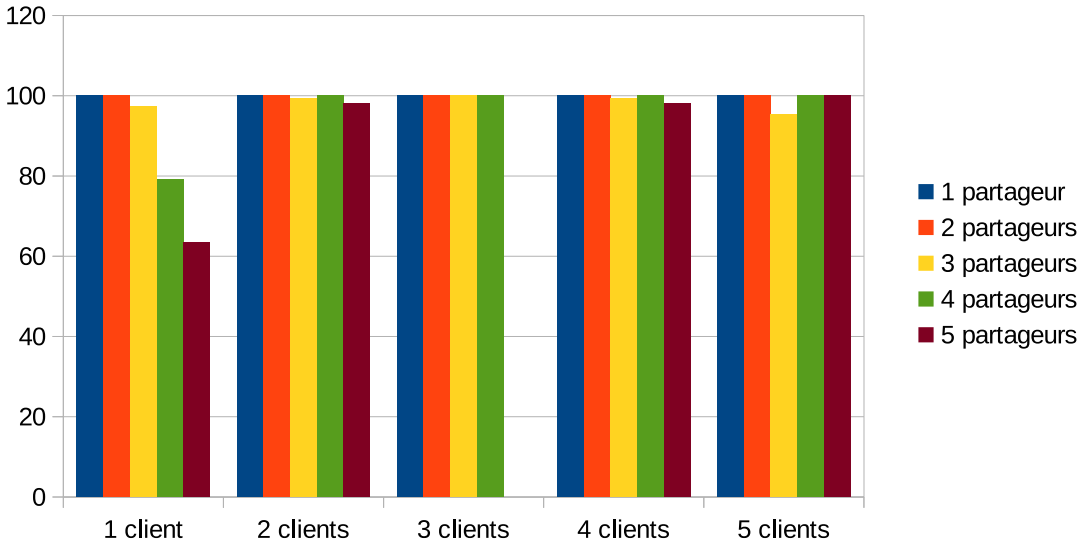

FIGURE 4.3 Rapport débit utilisable / débit disponible (en pourcentage)

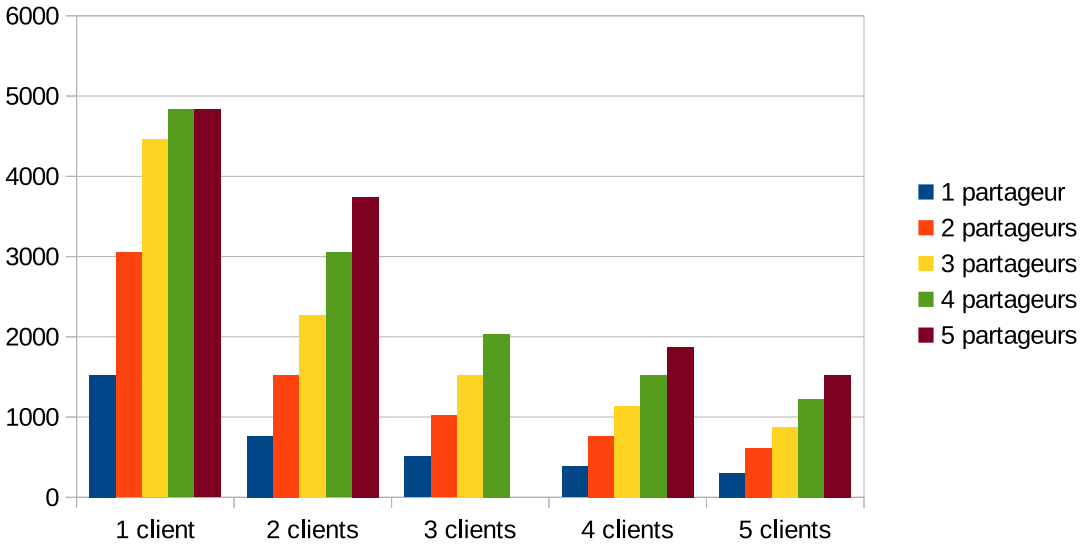

FIGURE 4.4 Débit de transfert effectif pour chaque client (en kbps)

Méta-heuristique

Comme nous avons pu le constater, nous arrivons très rapidement à une explosion combinatoire nous empêchant de parcourir l'ensemble de l'espace de recherche dans l'objectif de trouver la meilleure solution de distribution possible. Il s'agit en réalité d'un problème que nous pouvons ramener à un problème de sacs à dos multiples (*multiple Knapsack problem*) dont l'objectif est de minimiser le poids du sac le plus lourd. Ce problème est donc un problème de type NP-Complet. Afin de rester dans un temps raisonnable de calcul, nous n'avons donc pas d'autres choix que de faire appel à des heuristiques pour résoudre ce problème si le nombre de clients ou de partageurs augmentait. Il est cependant important de noter que l'optimisation de la répartition des différentes ressources sur plusieurs partageurs en fonction du nombre de clients n'est pas une opération qui doit s'effectuer en temps réel au moment où un client venait à faire une requête. Nous pouvons ainsi calculer préalablement les tableaux tel que nous l'avons fait pour l'ensemble des pages grâce à des heuristiques pour ensuite partager cette connaissance à tous les partageurs.

Nous avons donc proposé de concevoir une méta-heuristique de type tabou pour répondre à ce problème que nous avons développé avec le langage C dont les caractéristiques sont les suivantes :

— Fonction objectif : Minimiser dans un premier temps le volume d'octets transféré par le partageur le plus sollicité tout en minimisant le nombre de partageurs sollicités dans un second temps, l'objectif étant de paralléliser autant que possible les transferts sans soliciter inutilement plus de partageurs que nécessaire.

— Critère d'arrêt : Pour notre simulation nous avons mis un critère d'arrêt basé sur une non amélioration de notre solution courante depuis un certain nombre d'itérations (500). Dans le cadre d'un déploiement réel, il reviendra au partageur de référence de juger s'il préfère garder ce critère d'arrêt ou bien ou bien définir un temps fixe durant lequel l'heuristique est executée pour calculer les meilleurs tables de répartitions possibles afin de rendre son site partagé efficace.

— Longueur de la liste taboue : Nous avons fixé la longeur de manière arbitraire à 200 en gardant à l'esprit que plus la liste est longue plus nos chances d'obtenir le minimum global sont importantes mais augmente de manière importante notre temps de

caclcul. Comme nous n'avions pas de contrainte de temps, cela ne nous ne posait pas de problème. Dans un déploiement réel cependant, des contraintes de temps relative s'appliqueront puisque le partageur de référence ne sera peut-être pas près à calculer ses tables pendant plusieurs journées.

Après avoir développé et mis en fonctionnement notre heuristique avec les données de la page d'accueil de l'IETF, nous obtenenons ainsi les résultats présentés avec le tableau 4.8 lorsque nous nous comparons à un serveur à 100 Mbps.

TABLEAU 4.8 Résultats de l'execution d'une heuristique taboue pour le site de l'IETF

Type de machines	Nombre
Partageurs	71
Clients	19

Nos simulations d'heuristique nous donnent des résultats intéressants avec un nombre de partageurs raisonnable pour atteindre la capacité d'un serveur en salle machine. Si l'on a 71 partageurs et au moins 19 clients, les débits offerts pour chacun des clients seront des débits équivalents à un serveur connecté en 100 Mbps.

4.4 Disponibilité

Intéressons nous à présent à la disponibilité du modèle que nous proposons et comparons le au modèle traditionnel client/serveur. La disponibilité est la capacité pour un serveur de répondre à une demande. Les temps d'indisponibilité pour les serveurs sont principalement dus :

— Une coupure de courant au niveau du serveur.

— Une déconnexion du serveur du reste de l'Internet principalement dû à une coupure de la connexion Internet du fournisseur.

— Une déconnexion volontaire du fournisseur pour non-paiement du loyer mensuel de location du serveur ou pour des raisons de censure.

— Une mauvaise configuration par l'administrateur ou un piratage du serveur.

Nous mettons volontairement de côté une indisponibilité partielle entraînant des ralentis-

sements due à un trop grand nombre de requêtes simultanées pour plusieurs raisons. La première est que cette indisponibilité n'est que temporaire voire même n'est qu'un ralentissement et que par conséquent le serveur sera de nouveau capable sans intervention humaine de répondre de nouveau dans de meilleurs délais une fois le nombre de demandes retombées. La seconde est l'extrême difficulté de simuler ce type de comportement puisque ces derniers sont des conséquences directes de l'usage d'agents extérieurs peu prédictibles (l'usage des Internautes).

Disponibilité des serveurs en data-centre

À l'exception des serveurs d'entrée de gamme, la plupart des entreprises opérant des *data-centres* fournissent dans leur contrat de location des clauses quant à la disponibilité du service offert et s'engagent en cas de non respect à payer des indemnités à l'entreprise ou au particulier subissant le dommage. C'est ce que l'on appelle le *Service Level Agreement* (SLA). Bien évidemment, plus le pourcentage de disponibilité est important, plus le montant du loyer est important. Des SLA standards vont de 99,95% (soit 18 jours et 3 heures par année d'indisponibilité possible) à 99,999% (52 minutes) pour les modèles plus haut de gamme. Le fournisseur, pour assurer ces disponibilités, assurera une redondance maximale de toute l'infrastructure afin d'éviter les SPOF.

Disponibilité de notre modèle

Intéressons-nous à notre modèle à présent. Nous allons pour cela tenter de trouver si notre modèle peut être plus performant et si tel est le cas, dans quelle condition. Pour ce faire, nous allons définir quelques variables tout d'abord :

— S : correspondant au SLA du serveur en salle machine avec lequel nous nous comparons.

— n : correspondant au nombre de partageurs sur le réseau.

— p : correspondant à la probabilité pour un partageur d'être connecté. Statistiquement, cela correspond à une épreuve de Bernoulli de paramètre p dont le succès est définit par la capacité d'un partageur à être en mesure de répondre à une requête.

— X : correspondant à une variable aléatoire associée.

Nous partirons du principe que nos épreuves de Bernoulli sont indépendantes, autrement dit que la probabilité pour un partageur d'être connecté n'est pas affectée par celle d'autres partageurs. Nous pouvons donc appliquer une loi binomiale $X \hookrightarrow \mathrm{B}(n, p)$. La loi binômiale nous dit que :

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{4.2}$$

La probabilité que notre site partagé soit disponible est $P(X \geq 1)$, c'est à dire qu'au moins un partageur soit disponible pour répondre aux requêtes.

$$P(X \geq 1) = 1 - P(X = 0) \tag{4.3}$$

$$P(X \geq 1) = 1 - (\binom{n}{0} p^0 (1-p)^{n-0}) \tag{4.4}$$

$$P(X \geq 1) = 1 - (1-p)^n \tag{4.5}$$

Définissons $P(X \geq 1) = S$ afin de comparer les deux approches.

$$1 - (1-p)^n = S \tag{4.6}$$

$$n = log_{1-p}(1-S) \tag{4.7}$$

$$n = \frac{log(1-S)}{log(1-p)} \tag{4.8}$$

Comparons à présent notre modèle, à un modèle avec un SLA à 99,5%. Pour ce faire nous représenterons sous forme continue la fonction $f(p) = log(0,005)/log(1-p)$ représentée par la figure 4.5.

Faisons de même avec un SLA de 99,999% et définissons la fonction g tel que $g(p) = log(0,00001)/log(1-p)$ représentée par les figures 4.6 et 4.7.

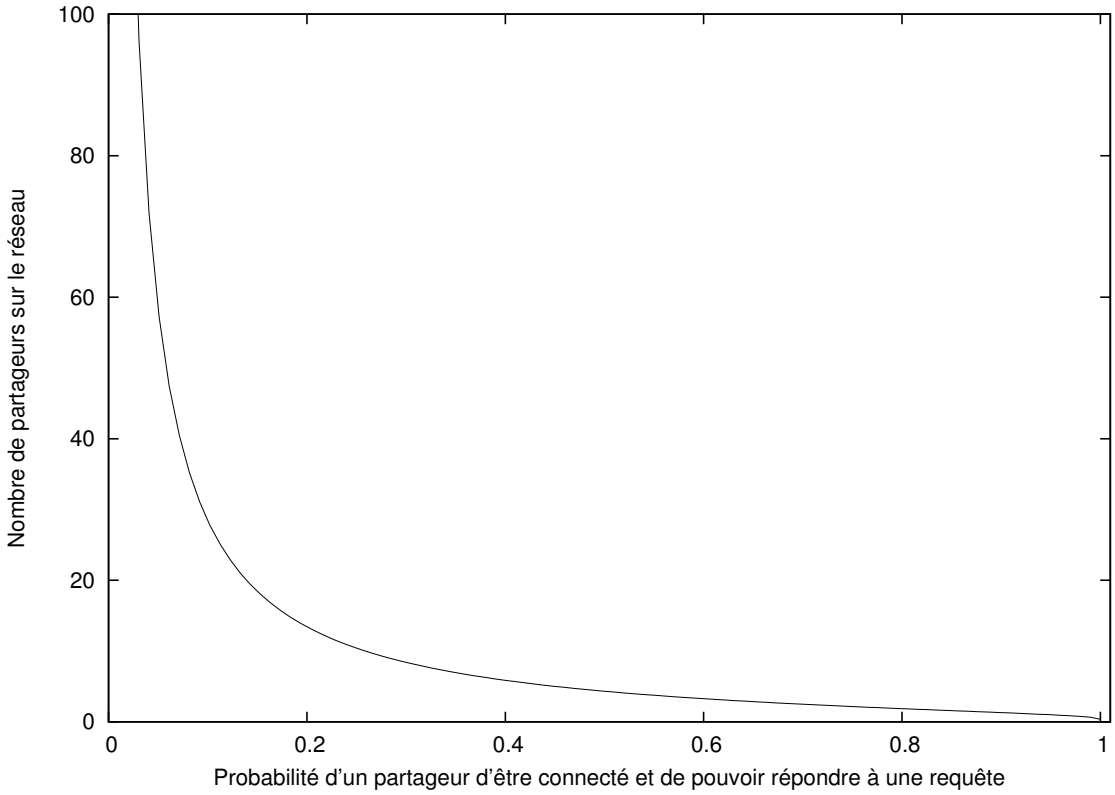

FIGURE 4.5 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,5%

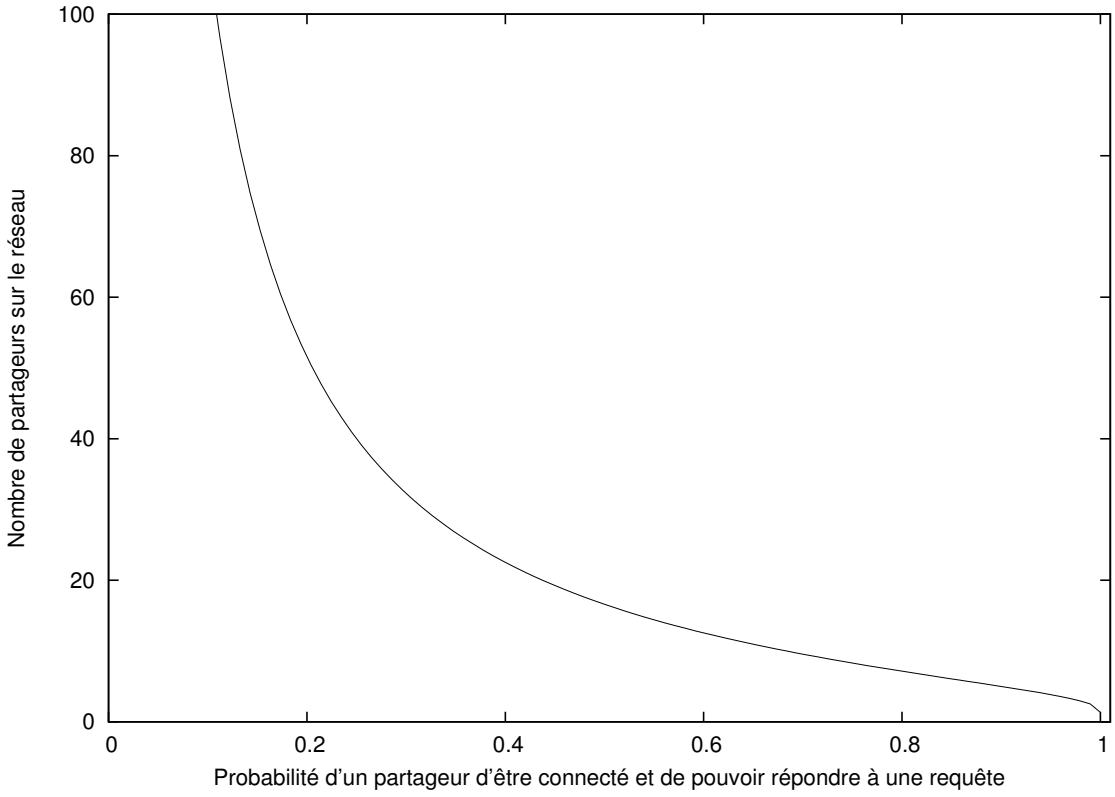

FIGURE 4.6 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,999%

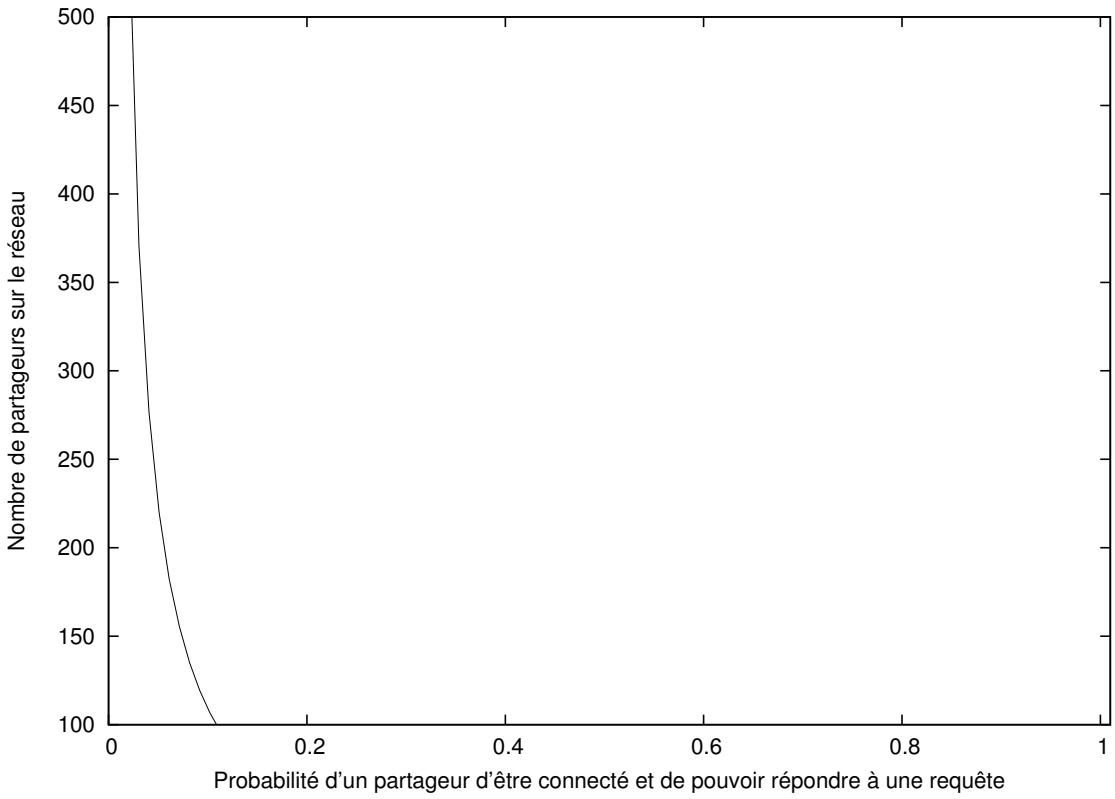

FIGURE 4.7 Comparaison entre le modèle proposé et un serveur avec un SLA à 99,999% (autre échelle)

Analyse des résultats

Comme nous pouvons le voir avec les graphiques précédents, notre modèle est tout à fait apte à remplacer le modèle client/serveur en matière de disponibilité. En effet, pour un SLA de 99,5%, nous avons besoin par exemple de plus de 50 partageurs joignable 10 % du temps pour atteindre le même niveau de fiabilité, ce qui devrait être relativement facile à atteindre pour un site assez populaire. Évidemment si le site est encore plus partagé, la disponibilité sera encore meilleure.

Pour un SLA de 99,999%, SLA de haute disponibilité assez rare et avec des tarifs de location souvent très dispendieux, le nombre de partageurs nécessaires est bien sûr plus important. La figure 4.6 étant à la même échelle que la figure 4.5, nous avons dû changer l'échelle pour absorber le nombre de partageurs nécessaires en gardant un taux de disponibilité par partageur de 10% avec la figure 4.7. Le nombre de partageurs ici à atteindre est de 110 pour atteindre un tel niveau, ce qui reste somme toute tout à fait raisonnable.

CHAPITRE 5

CONCLUSION

Dans le présent chapitre, nous repasserons en revue les points importants de notre projet de recherche auquel nous nous sommes confrontés.

Ce chapitre vient donc clore les démarches que nous avons mises en place pour mener à bien ce projet. Nous commencerons par faire une synthèse des travaux qui ont été accomplis pour par la suite présenter les limitations de notre solution. Pour finir, nous discuterons des améliorations possibles ainsi et que des travaux futurs qui pourront être développés.

5.1 Synthèse des travaux

La contribution majeure de notre travail a été de proposer un nouveau modèle permettant la décentralisation de serveurs web, traditionnellement utilisés en mode client/serveur. Notre modèle se distance cependant des autres projets puisqu'il se veut générique et en utilisant uniquement les standards du web que l'on trouve sur tous les clients, rendant notre approche totalement transparente pour eux. Pour ce faire, nous avons introduit la notion de partageur permettant à ce dernier de partager une partie de ces ressources pour le bien du site partagé en mettant à disposition une partie de sa bande passante et d'espace disque. Notre modèle s'appuie essentiellement sur le protocole DNS pour permettre la recherche d'un partageur capable de lui répondre, sans qu'il s'en aperçoive. Ainsi, de par notre modèle, nous tentons de jeter les bases d'un Internet où les hébergeurs web, application la plus populaire sur Internet, ne seraient pas concentrés et se rapprocheraient d'avantage d'un modèle pair-à-pair ou d'un modèle comme le fonctionnement de Usenet permettant de rétablir une symétrie entre les différents opérateurs Internet mais aussi en permettant la lutte contre la censure de par la diversivité des partageurs. Cela peut également être perçu comme une approche plus écologique, puisque tirant parti des ordinateurs déjà allumés et consommant de l'énergie quoi qu'il arrive. Comme les partageurs ne disposeront pour la plupart que d'un faible débit d'envoi, nous avons introduit des mécanismes, tant au niveau DNS (délégations de zones ou noms canoniques) que HTTP (`HTTP 302 FOUND` ou injections de ressources liées sur d'autres partageurs), afin de permettre autant que possible de répartir la charge du site web sur d'autres partageurs moins saturés. L'intérêt d'une telle approche est aussi de permettre une

meilleure évolutivité. Plus il y a des partageurs, plus notre réseau peut ainsi supporter de clients contrairement à l'approche client/serveur et c'est d'ailleurs ce que nous avons montré dans nos simulations.

5.2 Limitations de la solution proposée

Notre modèle, bien que fonctionnel, possède néanmoins quelques limitations. Nous l'avons vu dans la partie sur les mesures de performance, si le nombre de partageurs est relativement faible, notre modèle n'est pas du tout efficace dans les cas où il n'y a qu'un très faible nombre de partageurs ne disposant que de peu de ressources du site. Nous avons également dû faire des hypothèses sur des comportements humains qui affectaient nos résultats et avons essayé d'être le plus proche du comportement d'un utilisateur lambda. Il ne s'agit cependant que d'approximations et il serait donc intéressant de tester le modèle sur des ordinateurs personnels d'un nombre importants de personnes pour avoir des résultats plus précis.

5.3 Améliorations futures

Le modèle que nous avons proposé est une première approche et nécessite des améliorations qui pourraient faire l'objets de travaux futurs :

5.3.1 Sites dynamiques

Nous n'avons considéré dans notre approche que des sites statiques, mettant de côté une partie des sites proposant une interactivitié tels qu'on les trouve aujourd'hui. Il serait donc intéressant de proposer un système de mise-à-jour entre les différents partageurs lorsque l'un d'entre eux reçoit un contenu dynamique censé être répliqué sur les autres partageurs. Cela posera bien évidemment des problèmes de cohérence entre les différents partageurs et de nombreux travaux ont déjà été faits dans le domaine et pourrait être applicables.

5.3.2 Mode hybride

Nous pourrions également considérer une approche hybride à la fois en pair-à-pair et en client/serveur. Nous l'avons vu, les résultats ne sont pas très concluants lorsque le nombre de partageurs est faible et que de surcroît ils n'ont pas pour la plupart le contenu demandé par les clients. Il serait donc possible de permettre le partage du site par des partageurs mais de n'activer uniquement le site partagé au delà d'un certain seuil. Avant ce seuil, le site fonctionnerait de manière classique en mode client/serveur.

5.3.3 Limitation du volume de transferts

Certains opérateurs Internet facturent leurs abonnés sur le volume d'échange enregistré en débit montant et en débit descendant durant le mois. La facturation se fait habituellement selon le type d'abonnement choisi auquel est associé un volume de transfert compris dans le forfait. Au delà de ce volume, des frais additionnels s'appliqueront. Pour ce type d'abonnés, nous pourrions donc rajouter la possibilité de fixer un volume maximum de transfert après lequel le partageur cessera de faire partie du réseau jusqu'à ce que le compteur de volume soit réinitialisé par l'opérateur le mois suivant.

5.3.4 Sécurité

Permettre à n'importe quel personne de devenir partageur et de répondre aux clients à une partie des requêtes adresssées au site pose nécessairement des problèmes de sécurité. Le protocole HTTP ne possède aucun mécanisme de sécurité et est encore très utilisé sur Internet aujourd'hui. Cependant dans notre modèle, n'importe quel partageur a la possibilité de modifier les réponses qu'il envoie au client alors que dans les faits cela est plus difficile dans un mode client/serveur puisque le serveur est sous le contrôle de l'émetteur du contenu et qu'une modification malicieuse ne pourrait être possible que par un programme ou une personne malicieuse ayant accès au serveur ou bien lors de l'acheminement sur le réseau par des méthodes de *man-in-the-middle*. La solution serait donc certainement de mettre en place un système de signature numérique voire de chiffrement des communications afin d'en garantir l'intégrité. Le plus difficile sera cependant de le faire de manière transparante et donc d'utiliser HTTPS sans fournir la clef privée de chiffrement aux partageurs, seule possibilité pour un fureteur de vérifier une signature numérique et donc sans ajout d'add-on comme PGP.

5.3.5 Rapprocher les clients et partageurs

Toujours dans notre objectif d'améliorer le trafic entre opérateurs, il serait intéressant d'en profiter pour rapprocher topologiquement les clients des différents partageurs qui leurs envoient les ressources qu'ils souhaitent. En faisant cela, nous pourrions orienter les clients d'un opérateur Internet vers un partageur étant connecté sur le même opérateur. Cela permettrait de réduire de manière importante le trafic inter-AS, réduisant les factures des opérateurs, en particulier pour les opérateurs Tiers-3.

RÉFÉRENCES

BOARD, I. A. (2000). IAB Technical Comment on the Unique DNS Root. RFC 2826 (Informational).

CARLSSON, B. et GUSTAVSSON, R. (2001). The rise and fall of napster - an evolutionary approach. *Proceedings of the 6th International Computer Science Conference on Active Media Technology*. Springer-Verlag, London, UK, UK, AMT '01, 347–354.

CHEN, Z., GUO, S., YANG, Y. et ZHENG, K. (2007). Research of the apriority policy-based multi-hop bfs search algorithm in p2p network. *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*. 471–476.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P. et BERNERS-LEE, T. (1999). Hypertext transfer protocol – http/1.1.

GHAREED, M., ROUIBIA, S., PARREIN, B., RAAD, M. et THAREAU, C. (2013). P2PWeb : a Client/Server and P2P Hybrid Architecture for Content Delivery over Internet. *Proceedings of the Third International Conference on Communications and Information Technology ICCIT 2013*. Beirut, Liban, pp.1–5.

GROUP, W. W. W. (2012). Webrtc 1.0 : Real-time communication between browsers. `http://www.w3.org/TR/2012/WD-webrtc-20120821/`.

HAKIEM, N., SIDDIQI, M. et JAROT, S. (2012). Collision probability of one-to-many reversible mapping for ipv6 address generation. *Computer and Communication Engineering (ICCCE), 2012 International Conference on*. 599–602.

KUMAR, P., SRIDHAR, G. et SRIDHAR, V. (2005). Bandwidth and latency model for dht based peer-to-peer networks under variable churn. *Systems Communications, 2005. Proceedings*. 320–325.

LV, Q., CAO, P., COHEN, E., LI, K. et SHENKER, S. (2002). Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th international conference on Supercomputing*. ACM, New York, NY, USA, ICS '02, 84–95.

MAYMOUNKOV, P. et MAZIÈRES, D. (2002). Kademlia : A peer-to-peer information system based on the xor metric. *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, London, UK, UK, IPTPS '01, 53–65.

MOCKAPETRIS, P. (1987). Domain names - implementation and specification. RFC 1035 (INTERNET STANDARD).

NAKAMOTO, S. (2009). Bitcoin : A peer-to-peer electronic cash system.

REKHTER, Y., LI, T. et HARES, S. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard). Updated by RFCs 6286, 6608, 6793.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F. et BALAKRISHNAN, H. (2001). Chord : A scalable peer-to-peer lookup service for internet applications. *Proceedings of the ACM SIGCOMM '01 Conference*. San Diego, California.

TERRACE, J., LAIDLAW, H., LIU, H. E., STERN, S. et FREEDMAN, M. J. (2009). Bringing p2p to the web : security and privacy in the firecoral network. *Proceedings of the 8th international conference on Peer-to-peer systems*. USENIX Association, Berkeley, CA, USA, IPTPS'09, 7–7.

THOMSON, S., NARTEN, T. et JINMEI, T. (2007). IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard).

WU, J., LU, Z., LIU, B. et ZHANG, S. (2008). Peercdn : A novel p2p network assisted streaming content delivery network scheme. *CIT'08*. 601–606.